Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR – KI141502

IMPLEMENTASI *SECURE REAL-TIME TRANSPORT PROTOCOL* DENGAN MODE AES-OFB *ENCRYPTION* UNTUK PENGAMANAN *VOICE OVER INTERNET PROTOCOL*

I PUTU DWI PRATAMA ARIJAYA
NRP 5113100102

Dosen Pembimbing I
Henning Titi Ciptaningtyas, S.Kom., M.Kom.

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom., Ph.D.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

TUGAS AKHIR – KI141502

IMPLEMENTASI *SECURE REAL-TIME TRANSPORT PROTOCOL* DENGAN MODE AES-OFB *ENCRYPTION* UNTUK PENGAMANAN *VOICE OVER INTERNET PROTOCOL*

I PUTU DWI PRATAMA ARIJAYA
NRP 5113100102

Dosen Pembimbing I
Henning Titi Ciptaningtyas, S.Kom., M.Kom.

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom., Ph.D.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

[Halaman ini sengaja dikosongkan]

UNDERGRADUATE THESES – KI141502

IMPLEMENTATION OF SECURE REAL-TIME TRANSPORT PROTOCOL WITH AES-OFB ENCRYPTION MODE FOR SECURING VOICE OVER INTERNET PROTOCOL

I PUTU DWI PRATAMA ARIJAYA
NRP 5113100102

Supervisor I
Henning Titi Ciptaningtyas, S.Kom., M.Kom.

Supervisor II
Bagus Jati Santoso, S.Kom., Ph.D.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA 2017

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

IMPLEMENTASI *SECURE REAL-TIME TRANSPORT PROTOCOL* DENGAN MODE AES-OFB *ENCRYPTION* UNTUK PENGAMANAN *VOICE OVER INTERNET PROTOCOL*

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Rumpun Mata Kuliah Komputasi Berbasis Jaringan
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh
I PUTU DWI PRATAMA ARIJAYA
NRP : 5113 100 102

Disetujui oleh Dosen Pembimbing Tugas Akhir:

1. Henning Titi Ciptaningtyas, S.Kom., M.Kom. .....
   NIP: 19840708 201012 2 004 (Pembimbing 1)

2. Bagus Jati Santoso, S.Kom., Ph.D. .....
   NIP: - (Pembimbing 2)

SURABAYA
JUNI, 2017

[Halaman ini sengaja dikosongkan]

IMPLEMENTASI *SECURE REAL-TIME TRANSPORT PROTOCOL* DENGAN MODE AES-OFB *ENCRYPTION* UNTUK PENGAMANAN *VOICE OVER INTERNET PROTOCOL*

Nama Mahasiswa	**: I Putu Dwi Pratama Arijaya**
NRP	**: 5113100102**
Jurusan	**: Teknik Informatika FTIF-ITS**
Dosen Pembimbing 1	**: Henning Titi Ciptaningtyas, S.Kom., M.Kom.**
Dosen Pembimbing 2	**: Bagus Jati Santoso, S.Kom., Ph.D.**

Abstrak

Sebagai makhluk sosial, komunikasi merupakan hal yang penting dan sering dianggap kebutuhan yang harus dipenuhi oleh setiap manusia. Dengan perkembangan teknologi informasi dewasa ini, komunikasi dapat dilakukan dengan mudah. Salah satu teknologi yang banyak diminati adalah komunikasi berbasis Voice over Internet Protocol (VoIP). Untuk menjamin keamanan data saat dilakukan transmisi voice data, maka digunakan SRTP. Secure Real-Time Protocol (SRTP) adalah protokol yang dikembangkan dari Real-Time Transport Protocol (RTP) yang menyediakan fitur enkripsi, autentikasi dan integritas pesan, dan relay protection untuk RTP traffic dan untuk mengontrol traffic untuk RTP.

SRTP berfungsi untuk menjamin pengamanan pengiriman data khususnya multimedia file dengan enkripsi standar AES-SICM (Segmented Integer Counter Mode) dan AES-f8. Sedangkan mode cipher yang diajukan pada tugas akhir ini yaitu AES-OFB (Output Feedback) adalah blok cipher pada AES yang memungkinkan aliran cipher yang synchronous. OFB menghasilkan blok keystream (hasil enkripsi Initialization Vector (IV) dengan Key pada Block Cipher

Encryption AES) lalu dilakukan operasi XOR dengan plaintext untuk mendapatkan ciphertext.

Pengujian yang dilakukan menggunakan sebuah data audio dan di transmisikan dengan beberapajenis codec lalu dievaluasi aspek-aspek penunjang Quality of Service, penggunaan RAM, dan besar traffic pada jaringan. Diharapkan dengan implementasi AES-OFB pada SRTP dapat memenuhi kebutuhan komunikasi yang real-time dan aman. Setelah evaluasi didapatkan AES-OFB memiliki throughput yang baik, AES-OFB 128 memiliki kualitas layanan (QoS) terbaik pada durasi panggilan 50 detik menggunakan codec G711 dengan maksimum jitter 11.272 ms dan rata-rata jitter 2.536 ms sedangkan AES-OFB 256 memiliki kualitas layanan (QoS) terbaik pada durasi panggilan 200 detik menggunakan codec G722 dengan maksimum jitter 17.398 ms dan rata-rata jitter 1.912 ms.

Kata kunci: VoIP, SRTP, AES-OFB, Quality of Service, Jitter.

IMPLEMENTATION OF SECURE REAL-TIME TRANSPORT PROTOCOL WITH AES-OFB ENCRYPTION MODE FOR SECURING VOICE OVER INTERNET PROTOCOL

Nama Mahasiswa	**: I Putu Dwi Pratama Arijaya**
NRP	**: 5113100102**
Jurusan	**: Teknik Informatika FTIF-ITS**
Dosen Pembimbing 1	**: Henning Titi Ciptaningtyas, S.Kom., M.Kom.**
Dosen Pembimbing 2	**: Bagus Jati Santoso, S.Kom., Ph.D.**

Abstract

As a social being, communication is an important and usually considered as a needs that must be met by humans. With the development of information technology today, communication can be done easily. One of the most popular technologies is communication based on Voice over Internet Protocol (VoIP). To guarantee the data security while voice data transmission, so SRTP is being used. Secure Real-Time Protocol (SRTP) is a protocol that developed from Real-Time Protocol (RTP) profile that provide encryption, authentication, and message integrity feature, and also relay protection for RTP traffic and controlling RTP traffic.

SRTP used to guarantee the security of the data transmission especially multimedia file with encryption standard AES-SICM (Segmented Integer Counter Mode) dan AES-f8. While cipher mode that proposed in this thesis is AES-OFB (Output Feedback). AES-OFB is a block cipher in AES that allows synchronous cipher flow. OFB generates keystream block (encryption result of Initialization Vector (IV) with key in Block Cipher Encryption AES) then do XOR operation with plaintext to produce the ciphertext.

The test use one type of audio that transmitted with different type of codec in SRTP to evaluate the Quality of Service aspecs, memory used, and traffic size in network. After the evaluation of the testing data AES-OFB has good throughput, AES-OFB 128 has the best quality of service in 50 seconds of call duration using G711 codec with maximum jitter 11.272 ms and mean jitter 2.536 ms while AES-OFB 256 has the best quality of service in 200 seconds of call using G722 codec with maximum jitter 17.398 ms and mean jitter 1.912 ms.

Keywords: VoIP, SRTP, AES-OFB, Quality of Service, Jitter.

KATA PENGANTAR

Segala puji syukur kepada Ida Sang Hyang Widhi Wasa, yang telah melimpahkan anugerah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul ***"*IMPLEMENTASI *SECURE REAL-TIME TRANSPORT PROTOCOL* DENGAN MODE AES-OFB *ENCRYPTION* UNTUK PENGAMANAN *VOICE OVER INTERNET PROTOCOL"*.** Dengan pengerjaan Tugas Akhir ini, penulis bisa belajar lebih banyak untuk memperdalam dan meningkatkan apa yang telah didapatkan penulis selama menempuh perkuliahan di Teknik Informatika ITS. Dengan Tugas Akhir ini penulis juga dapat menghasilkan suatu implementasi dari apa yang telah penulis pelajari.

Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. Ida Sang Hyang Widhi Wasa.
2. Keluarga di Bali khususnya Ibu, Bapak, Nida, Ryan, Tante Alit, dan Om Guna yang telah memberikan dukungan moral dan material serta do'a yang tak terhingga untuk penulis. Serta selalu memberikan semangat dan motivasi pada penulis dalam mengerjakan Tugas Akhir ini.
3. Keluarga di Surabaya Om Adi dan Tante Puji yang selalu memberi semangat dan dukungan selama penulis kuliah di ITS Surabaya.
4. Ibu Henning Titi Ciptaningtyas, S.Kom., M.Kom. selaku pembimbing 1 yang telah membantu, membimbing, dan memotivasi penulis dalam menyelesaikan Tugas Akhir ini dengan sabar.
5. Bapak Bagus Jati Santoso, S.Kom., Ph.D. selaku pembimbing 2 yang juga telah membantu, membimbing, dan memotivasi penulis dalam menyelesaikan Tugas Akhir ini dengan sabar.

6. Bapak Waskitho Wibisono, S.Kom., M.Eng., Ph.D. selaku dosen wali yang telah membimbing penulis selama 3.5 tahun berkuliah di Teknik Informatika ITS.
7. Bapak Dr. Darlis Herumurti, S.Kom., M.Kom. selaku Kepala Jurusan Teknik Informatika ITS.
8. Bapak Radityo Anggoro, S.Kom., M.Sc. selaku koordinator Tugas Akhir, dan segenap dosen Teknik Informatika yang telah memberikan ilmunya.
9. Sahabat sehidup, semati, sepenanggungan Dika, Ray, dan Dicky.
10. Teman-teman kontrakan UWUS mas Indra, mas Wicak, mas Dewak, mas Semara, mas Kusnanta, mas Nyoman, dan mas Nanda yang selalu memberi semangat.
11. Teman-teman kos U69 Gara, Gembok, Dwiko, Dewangga, Abek, Adnan, Firdan, dan Ipoy yang membuat tahun pertama penulis berkuliah terasa istimewa.
12. Teman-teman angkatan 2013 yang yang telah berbagi ilmu, dan memberi motivasi kepada penulis.
13. Serta semua pihak yang yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depannya.

Surabaya, Juni 2017

DAFTAR ISI

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

DAFTAR KODE SUMBER

[Halaman ini sengaja dikosongkan]

DAFTAR PSEUDOCODE

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

1.1 Latar Belakang

Sebagai makhluk sosial, komunikasi merupakan hal yang penting dan sering dianggap kebutuhan yang harus dipenuhi oleh setiap manusia. Dengan perkembangan teknologi informasi dewasa ini, komunikasi dapat dilakukan dengan mudah. Jarak dan waktu bukan menjadi halangan yang besar di dunia yang serba *online* seperti sekarang. Tingginya permintaan terhadap media komunikasi *online* membuat banyak pengembang mulai mengembangkan sistem yang mampu menjawab kebutuhan banyak orang. Salah satunya layanan komunikasi berbasis VoIP (*Voice over Internet Protocol*) dan juga beberapa aplikasi yang mendukung *video call*.

Saat ini, pengamanan transmisi di perangkat multimedia menjadi isu yang banyak diperhitungkan seiring makin populernya media yang menyediakan komunikasi *real-time*. Hal ini tidak mudah, karena semakin rumit pengamanan yang dilakukan maka akan semakin banyak komputasi yang dilakukan dan akan berisiko mengurangi kualitas layanan (QoS), begitu juga sebaliknya. Untuk mencapai kualitas layanan yang baik maka pemilihan algoritma untuk pengamanan tidak boleh sembarangan. Algoritma enkripsi harus mendukung beberapa aspek dalam transmisi di multimedia seperti ukuran data yang besar, *bandwidth* yang tinggi, dan kebutuhan *real-time* lain. Maka algoritma enkripsi ini diharapkan dapat menjawab tantangan-tantangan seperti proses dalam kecepatan tinggi, tingkat keamanan yang tinggi, dan kualitas layanan yang baik.

Salah satu metode yang bisa digunakan adalah skema streaming dengan menggunakan *Secure Real-Time Transport Protocol* dengan kualitas layanan yang tinggi tanpa mengurangi sisi sekuritas. Skema juga bisa disebut dengan SRTP [1]. Skema SRTP ini juga dapat digabungkan dengan mekanisme enkripsi

yang lain. Mekanisme yang sering digunakan adalah Perkalian XOR, Transposisi, DES dan AES. Hal ini membuat SRTP banyak dipakai untuk oleh pengembang dalam meningkatkan keamanan saat transmisi data.

Dalam tugas akhir ini, akan digunakan skema *streaming* berdasarkan modifikasi SRTP dengan mode AES-OFB *encryption*. Sehingga saat transmisi berlangsung, data transmisi berupa *voice data* dapat di enkripsi. Dengan metode ini, data-data yang sebelumnya dikirim secara langsung memiliki tingkat keamanan yang tinggi. Dan data enkripsi tersebut dapat di dekripsi kembali dengan kecepatan yang tinggi sehingga diharapkan tidak mengurangi kualitas layanan dari *live streaming*.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana pengaruh mode enkripsi terhadap besar paket RTP pada VoIP?
2. Bagaimana efek implementasi mode enkripsi AES-OFB terhadap penggunaan RAM pada saat panggilan VoIP?
3. Bagaimana efek implementasi mode enkripsi AES-OFB pada SRTP terhadap nilai *throughput* pada VoIP?
4. Bagaimana efek implementasi mode enkripsi AES-OFB pada SRTP terhadap nilai *jitter* pada VoIP?
5. Bagaimana efek implementasi mode enkripsi AES-OFB pada SRTP terhadap kualitas layanan (*Quality of Service*) pada VoIP?

1.3 Batasan Masalah

Permasalahan yang dibahas dalam tugas akhir ini memiliki beberapa batasan antara lain:

1. Menggunakan *library* Libsrtp dalam bahasa pemrograman C.

2. Pengujian dilakukan dalam jaringan lokal Teknik Informatika FTIf ITS.
3. Pengujian menggunakan dua buah klien SIP dalam sebuah sesi panggilan VoIP.
4. Menggunakan metode SRTP untuk mengubah data transmisi suara.
5. Mengimplementasikan AES-OFB sebagai mode *cipher* pada AES-ICM.
6. Menggunakan *User Datagram Protocol* (UDP) sebagai protokol media.

1.4 Tujuan

Tujuan dari pembuatan tugas akhir ini untuk mengamanakann *voice data* saat transmisi (*streaming*) dan mendapatkan kualitas layanan (*Quality of Service*) yang baik.

1.5 Manfaat

Manfaat dari hasil pembuatan tugas akhir ini yaitu pengguna merasa aman saat melakukan komunikasi menggunakan *voice call* melalui VoIP.

1.6 Metodologi

Tahapan-tahapan yang dilakukan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir.
   Tahap awal untuk memulai pengerjaan Tugas Akhir adalah penyusunan proposal Tugas Akhir. Penyusunan proposal Tugas Akhir dilaksanakan untuk merumuskan masalah serta melakukan penetapan rancangan dasar dari sistem yang akan dikembangkan dalam pelaksanaan Tugas Akhir ini.

2. Studi literatur

   Pada tahap ini dilakukan pemahaman informasi dan literatur yang diperlukan untuk tahap implementasi program. Tahap ini diperlukan untuk membantu memahami penggunaan komponen-komponen terkait dengan sistem yang akan dibangun, antara lain VoIP, metode SRTP, enkripsi AES-OFB, dan *library* libsrtp.

3. Analisis dan perancangan perangkat lunak

   Tahap ini meliputi perancangan sistem berdasarkan studi literatur dan pembelajaran konsep teknologi dari perangkat lunak yang ada. Tahap ini mendefinisikan alur dari implementasi. Langkah-langkah yang dikerjakan juga didefinisikan pada tahap ini. Pada tahapan ini dibuat *prototype* sistem, yang merupakan rancangan dasar dari sistem yang akan dibuat. Serta dilakukan desain fungsi yang akan dibuat yang ditunjukkan melalui *pseudocode*.

4. Implementasi perangkat lunak

   Implementasi perangkat lunak merupakan tahap membangun rancangan program yang telah dibuat. Pada tahap ini akan direalisasikan mengenai rancangan apa saja yang telah didefiniskan pada tahap sebelumnya. Fungsi yang ada pada tahap ini merupakan fungsi hasil implementasi dari tahap analisis dan perancangan perangkat lunak.

5. Pengujian dan evaluasi

   Pada tahap ini dilakukan uji coba pada data yang telah dikumpulkan. Tahap ini digunakan untuk mengevaluasi kinerja program serta mencari masalah yang mungkin timbul saat program dievaluasi serta melakukan perbaikan jika terdapat kesalahan pada program.

6. Penyusunan buku Tugas Akhir

   Pada tahap ini disusun buku yang memuat dokumentasi mengenai perancangan, pembuatan serta hasil dari implementasi perangkat lunak yang telah dibuat.

1.7 Sistematika Penulisan Laporan Tugas Akhir

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir ini. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini:

Bab I Pendahuluan

Bab yang berisi mengenai latar belakang, tujuan, dan manfaat dari pembuatan Tugas Akhir. Selain itu perumusan masalah, batasan masalah, metodologi yang digunakan, dan sistematika penulisan juga merupakan bagian dari bab ini.

Bab II Tinjauan Pustaka

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan Tugas Akhir ini.

Bab III Analisis dan Perancangan Perangkat Lunak

Bab ini berisi tentang dasar dari algoritma yang akan diimplementasikan pada Tugas Akhir ini.

Bab IV Implementasi

Bab ini membahas mengenai implementasi dari rancangan yang telah dibuat pada bab sebelumnya.

Bab V Uji Coba Dan Evaluasi

Bab ini menjelaskan mengenai kemampuan perangkat lunak dengan melakukan pengujian kebenaran dan

pengujian kinerja dari perangkat lunak yang telah dibuat sesuai dengan data yang diujikan.

Bab VI Kesimpulan Dan Saran

Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang telah dilakukan dan saran untuk pengembangan perangkat lunak ke depannya.

BAB II
TINJAUAN PUSTAKA

Bab ini berisi penjelasan teori-teori yang berkaitan dengan algoritma yang diajukan pada pengimplementasian program. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap program yang dibuat dan berguna sebagai penunjang dalam pengembangan perangkat lunak.

2.1 Voice over Internet Protocol

Voice over Internet Protocol atau yang dikenal dengan VoIP adalah teknologi yang mampu mengirimkan data suara, video dan data yang berbentuk paket secara *real-time* dengan jaringan yang menggunakan *Internet Protocol* (IP) [2] [3]. Teknologi VoIP bekerja dengan cara mengubah suara yang merupakan sinyal analog menjadi sinyal digital yang dapat dikirimkan melalui jaringan yang memanfaatkan IP. Setelah diubah menjadi sinyal digital, kemudian ditranslasikan ke dalam paket-paket IP yang kemudian ditransmisikan melalui jaringan.

Tujuan pengimplementasian VoIP adalah untuk menekan biaya instansi (perusahaan, sekolah, rumah sakit, dll.) maupun individu dalam melakukan komunikasi jarak dekat maupun jarak jauh (interlokal/ SLI). Penekanan biaya itu dapat dilakukan dengan cara memanfaatkan jaringan data yang sudah ada. Sehingga apabila ingin membuat jaringan telekomunikasi VoIP tidak perlu membangun infrastruktur baru yang biasanya memerlukan biaya yang besar. VoIP dalam penerapannya menggunakan sistem jaringan LAN dan didukung protokol-protokol VoIP. Beberapa standarisasi protokol komunikasi pada teknologi VoIP adalah SIP (*Session Initiation Protocol*) dan IAX2 (*Internet Asterisk eXchange* 2).

2.2 Session Initiation Protocol

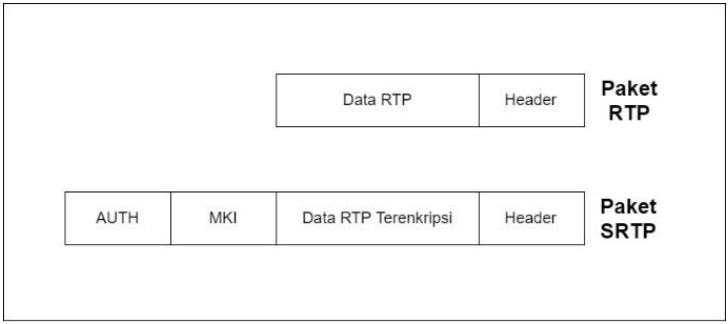
Session Initiation Protocol (SIP) [4] merupakan salah satu standar pensinyalan dan pengontrolan sesi dari *packet telephony* yang dikembangkan oleh IETF sebagai bagian dari *Internet Multimedia Conferencing Architechture*. SIP merupakan sebuah *application-layer protocol* untuk membentuk, memodifikasi, dan menterminasi sebuah sesi multimedia. Sesi multimedia adalah pertukaran data antar pengguna yang bisa meliputi suara, video, dan text. Seperti layaknya HTTP, SIP merupakan protokol *client-server* yang menggunakan model transaksi *request* dan *response*.

SIP menggunakan struktur protokol yang sederhana, sehingga operasinya cepat dan fleksibel. SIP tidak menyediakan layanan secara langsung, tetapi menyediakan pondasi yang dapat digunakan oleh protokol aplikasi lainnya untuk memberikan layanan yang lebih lengkap bagi pengguna, misalnya dengan RTP (*Real Time Transport Protocol*) untuk transfer data secara *real-time*, dengan SDP (*Session Description Protocol*) untuk mendiskripsikan sesi multimedia, dengan MEGACO (*Media Gateway Control Protocol*) untuk komunikasi dengan PSTN (*Public Switch Telephone Network*).

2.3 Secure Real-Time Transport Protocol

Secure Real-Time Transport Protocol (SRTP) [1] adalah metode yang dikembangkan oleh tim gabungan antara CISCO dan Ericson yang terdiri dari pada ahli di bidang *Internet Protocol* (IP) dan kriptografi. SRTP pertama kali dipublikasikan pada Maret 2004 oleh Internet Engineering Task Force (IETF) sebagai RFC3711. SRTP sendiri dikembangkan dari *Real-Time Transport Protocol* (RTP) yang menyediakan fitur enkripsi, autentikasi dan integritas pesan, dan *relay protection* pada lalu lintas data RTP dan mengontrol *traffic* untuk RTP pada aplikasi *unicast* (aplikasi yang berjalan dari satu pengguna ke satu pengguna lain) dan *multicast* (aplikasi yang berjalan dari satu pengguna ke banyak pengguna).

Perbedaan paket RTP dan paket SRTP dapat dilihat pada Gambar 2.1.



Gambar 2.1 Perbedaan paket RTP dan SRTP

Bagian AUTH pada paket SRTP menjadi penanda autentikasi pada data-data yang membawa pesan terautentikasi jadi pengirim data/pesan dapat mengenkripsi data terlebih dahulu sebelum proses autentikasi dijalankan. Bagian MKI (*Master Key Indentifier*) mengidentifikasi kunci master diturunkan kunci sesi mana untuk mengenkripsi data/pesan. Secara umum SRTP berfungsi untuk menjamin pengamanan dari pengiriman data file multimedia, *voice over internet protocol* (VoIP) dan meminimalisir ancaman *Denial of Service* (DoS). SRTP menggunakan algoritma AES (*Advanced Encryption Standard*) sebagai metode enkripsi dalam pengiriman data. Pada aplikasinya SRTP memiliki 2 buah mode, yaitu *Segmented Integer Counter*, dan AES di f8-mode. Selain itu SRTP juga dapat berjalan dengan mode *null cipher*, dimana pengiriman data tidak dilindungi dengan algoritma enkripsi.

2.4 Kualitas Layanan

Kualitas layanan (QoS) adalah sebuah cara untuk mengukur kualitas sebuah jaringan pada saat melakukan pengiriman data seperti video beresolusi tinggi dan data yang bersifat *real-time* [5]. Beberapa kualitas layanan yang dapat diukur adalah *data delay*, *packet drop* yaitu berapa banyak paket data yang di *drop* saat transmisi, *jitter* (variasi *delay*) yaitu perbedaan *delay* dari tiap *packet* untuk sampai ke tujuan (*client*), throughput yaitu dan *latency* yaitu lama waktu yang dibutuhkan oleh *packet* untuk sampai ke *client*.

2.5 Audio *Codec* (*Code/Decode*)

Codec adalah proses kompresi dan dekompresi data analog menjadi digital dan juga sebaliknya yang dilakukan oleh sebuah program komputer untuk keperluan transmisi data pada jaringan dan dapat diterjemahkan saat telah sampai pada tujuan. Dalam industri telekomunikasi, data yang ditransmisikan melalui *codec* adalah paket data pada saat panggilan berlangsung. Dalam melakukan panggilan, *codec* digunakan sebagai mekanisme untuk memaksimalkan pemanfaatan *bandwidth* dengan mengompresi data panggilan untuk kemudian didekompres saat data telah sampai.

Beberapa jenis audio *codec* yang digunakan pada tugas akhir ini adalah GSM (Global System for Mobile Communications) yaitu standar dari ETSI (European Telecommunications Standards Institute) [6], G722 [7] dan G711 [8] yaitu standar dari ITU-T (International Telecommunication Union – Telecomunication Standarization Section). Spesifikasi dari tiap *codec* yang digunakan dalam tugas akhir ini dapat dilihat pada Tabel 2.1.

Tabel 2.1 Bandwidth Codec GSM, G722, dan G711

No	Jenis *Codec*	Standar	*Bandwidth* (kbps)	*Bandwidth* (kBps)
1	GSM	ETSI	13	1.625
2	G722	ITU-T	64	8
3	G711	ITU-T	64	8

2.6 Advanced Encryption Standard

Advanced Encryption Standard (AES) [9] adalah algoritma enkripsi yang digunakan untuk melindungi data elektronik. Algoritma dari AES merupakan blok *cipher* yang simetris yang dapat melakukan enkripsi dan dekripsi pada informasi/data. Data yang diproses pada AES memiliki panjang 128-*bit* dan dapat diproses menggunakan tiga macam kunci yaitu kunci sepanjang 128-*bit* atau dikenal dengan AES-128, 192-*bit* atau dikenal dengan AES-192, dan 256-*bit* atau dikenal dengan AES-256.

Algoritma AES terdiri dari beberapa langkah yaitu: ekspansi kunci, langkah inisasi, langkah perulangan, dan langkah akhir. Ekspansi kunci dari AES terdiri dari tiga langkah yaitu rotasi, operasi menukar susunan *word* sebagai contoh susunan *word* 32-bit "1D 2C 3A 4F" menjadi "2C 3A 4F 1D", rcon yaitu bentuk dari eksponensiasi dari 2 yang digunakan pada dokumentasi Rijndael, dalam bentuk polinomial angka 2 adalah $2 = 00000010 = 0x^7 + 0x^6 + 0x^5 + 0x^4 + 0x^3 + 0x^4 + 0x^3 + 0x^2 + 1x + 0 = b$, maka dirumuskan $\text{rcon}(i) = b^{i-1}$ untuk $i = 8$ atau lebih lengkapnya $\text{rcon}(i) = b^{i-1} \bmod x^8 + x^4 + x^3 + x + 1$ untuk nilai $i = x$ sebagai contoh rcon(1) = 0x01, rcon(2) = 0x02, rcon(9) = 0x1b, dan s-box yaitu operasi menukar isi dari bit awal dengan isian baru pada s-box yang berasal dari koordinat bit awal. Terdapat tiga macam perulangan dalam ekspansi kunci tergantung panjang kunci dari AES (10 siklus perulangan untuk kunci 128-*bit*, 12 siklus perulangan untuk kunci 192-*bit*, dan 14 siklus perulangan untuk kunci 256-*bit*).

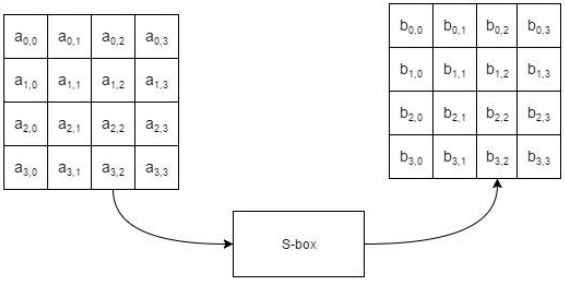
Selanjutnya langkah inisiasi atau AddRoundKey pertama yaitu setiap *byte* dari blok *cipher* awal dikombinasikan dengan blok kunci ronde dengan operasi *bitwise* XOR. Lalu dilanjutkan dengan

langkah perulangan yang dimulai dengan langkah SubBytes yaitu setiap *byte* $a_{i,j}$ dalam matriks keadaan digantikan oleh SubBytes $S(a_{i,j})$ menggunakan 8-bit kotak substitusi Rijndael S-box. Rijndael S-box adalah sebuah matriks substitusi berukuran 16 x 16 yang direpresentasikan dalam heksadesimal yang dapat dilihat pada Tabel 2.2.

Tabel 2.2 Rijndael S-box

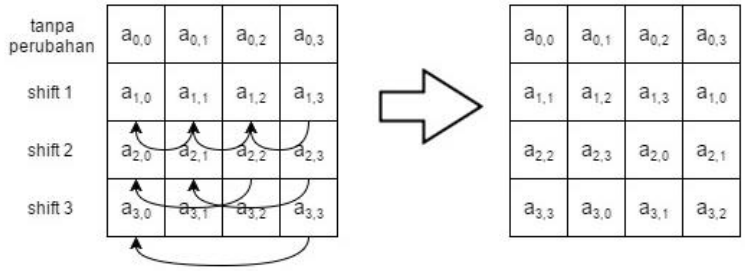
		y															
		00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**0a**	**0b**	**0c**	**0d**	**0e**	**0f**
x	**00**	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Setiap bit pada matriks awal $a_{i,j}$ akan dibagi menjadi dua bagian yaitu bagian i sebagai sumbu x dan bagian j sebagai sumbu y. sebagai contoh isi dari matriks awal adalah 19 maka i = 1 dan j = 9, setelah disubstitusi dengan Rijndael S-box menjadi d4 dan begitu seterusnya hingga seluruh matriks tersubstitusi.



Gambar 2.2 Langkah SubBytes

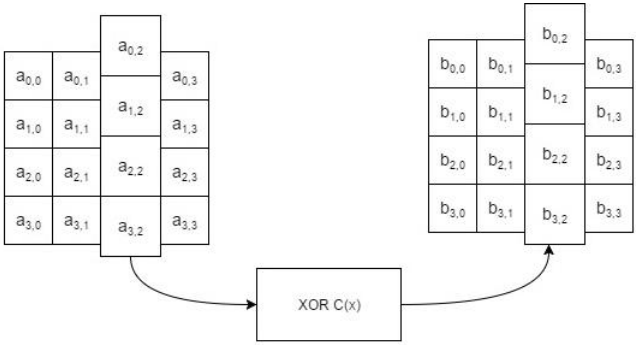
Dilanjutkan dengan langkah SwiftRow yaitu operasi menggeser pada setiap elemen blok yang dilakukan per barisnya. Untuk baris pertama tidak dilakukan operasi pergeseran blok, baris kedua dilakukan pergeseran 1 *byte*, baris ketiga dilakukan pergeseran 2 *byte*, dan baris keempat dilakukan pergeseran 3 *byte*. Pergeseran yang dilakukan adalah pergeresan ke kiri dan apabila melewati batas kiri blok maka kembali ke posisi kanan dari blok tersebut.



Gambar 2.3 Langkah SwiftRow

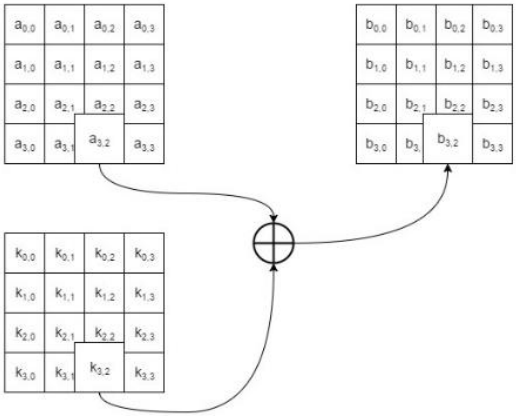
Selanjutnya blok *cipher* dari langkah SwiftRow ini menuju ke langkah MixColumn. Pada langkah MixColumn dilakukan operasi perkalian *bitwise* XOR tiap elemn dari blok *cipher* dengan matriks yang telah ditentukan sebelumnya. Ukuran dari matriks ini mengikuti besar dari blok *cipher*. Perkalian yang dilakukan seperti perkalian *dot product* matriks pada umumnya lalu hasilnya dimasukkan pada blok *cipher* yang baru. Langkah MixColumn dapat dilihat pada Gambar 2.4.

Langkah MixColumn ini tidak dilakukan sebanyak siklus perulangan atau hanya dilakukan sebanyak n-1 (n = 10 untuk 128-bit, n = 12 untuk 192-bit, dan n = 14 untuk 256-bit) dikarenakan pada awal proses AES dilakukan langkah AddRoundKey sehingga pada akhir proses enkripsi AES tidak dilakukan langkah MixColumn agar sesuai dengan langkah dekripsinya.

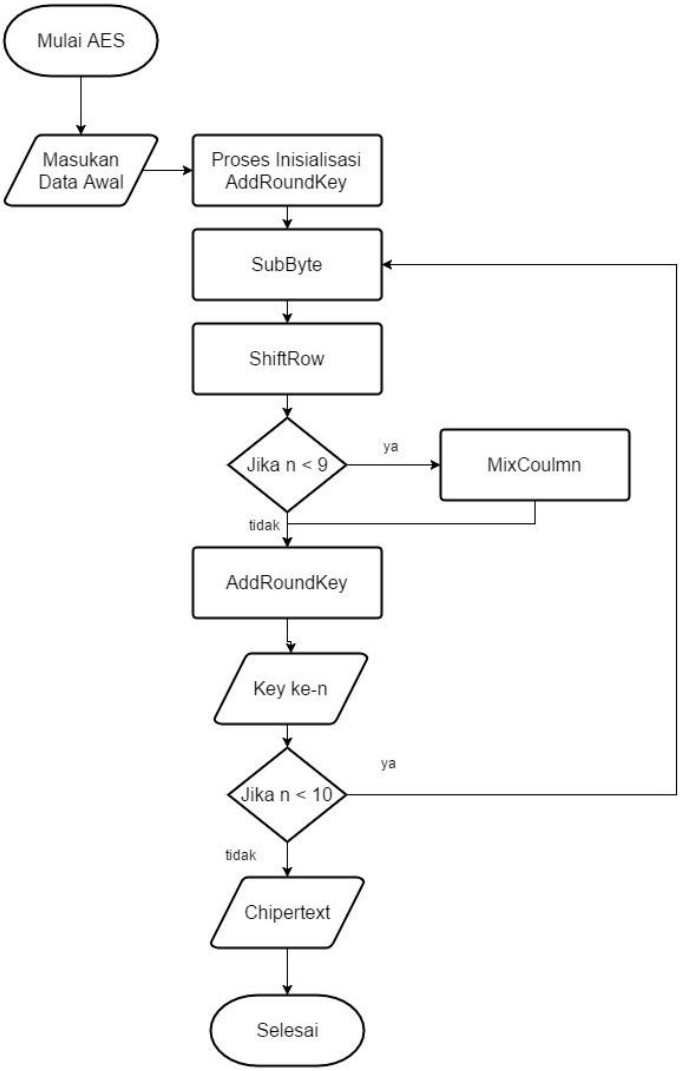


Gambar 2.4 Langkah MixColumn

Setelah langkah MixColumn dilakukan maka selanjutnya kembali ke langkah AddRoundKey seperti pada langkah inisiasi dengan setiap blok pada kunci ronde dikalikan dengan blok *cipher* saat ini menggunakan *bitwise* XOR menghasilkan kunci ronde yang baru. Langkah AddRoundKey dapat dilihat pada Gambar 2.5. Keseluruhan dari jalan AES dapat dilihat pada Gambar 2.6.



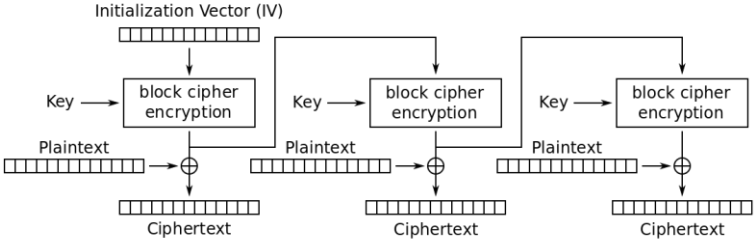
Gambar 2.5 Langkah AddRoundKey



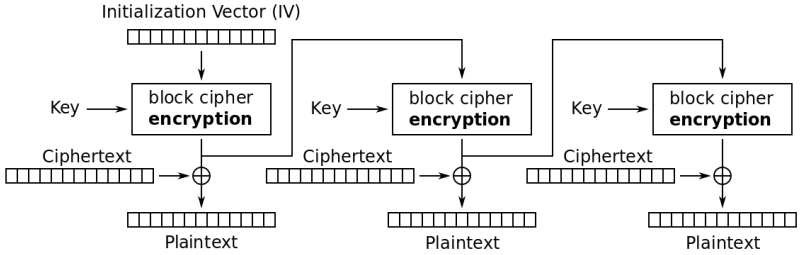
Gambar 2.6 Diagram Alir AES

2.7 AES Output Feedback (OFB)

Output Feedback (OFB) [10] merupakan blok *cipher* pada AES yang memungkinkan aliran *cipher* yang *synchronous*. OFB menghasilkan blok *keystream* (hasil enkripsi *Initialization Vector* (IV) dengan *Key* pada enkripsi blok *cipher* AES) lalu dilakukan operasi *bitwise* XOR dengan *plaintext* untuk mendapatkan *ciphertext*. Sama seperti blok *cipher* lainnya, kelebihan *bit plaintext* akan diproses pada blok *cipher* menggunakan *keystream* sebelumnya menjadi IV pada proses enkripsi berikutnya. Karena operasi XOR yang simetris maka proses enkripsi dan dekripsi sama sepeti yang ditunjukkan pada Gambar 2.7 dan Gambar 2.8.



Gambar 2.7 Enkripsi Output Feedback (OFB)



Gambar 2.8 Dekripsi Output Feedback (OFB)

Operasi blok *cipher output feedback* dimulai dengan menentukan IV (*Initiatization Vector*) yang dibuat secara acak. Dalam blok *cipher* lainnya, data yang diproses dalam blok AES adalah data yang akan di enkripsi atau data *plaintext*, namun pada blok *cipher* OFB yang masuk ke blok AES adalah IV. Dalam proses AES juga dibuat kunci ronde awal sebagai langkah inisialisasi AES. Setelah seluruh rangkaian AES selesai dilakukan, hasil dari proses AES dan data *plaintext* dikombinasikan menggunakan operasi *bitwise* XOR. Apabila belum seluruh *plaintext* yang diproses *bitwise* XOR atau baru 32-bit awal dari *plaintext* yang diproses, maka operasi OFB melanjutkan proses enkripsi untuk bit-bit pada data *plaintext* yang belum dienkripsi.

Untuk blok *cipher* selanjutnya membutuhkan masukan IV seperti langkah pertama. Tidak seperti langkah pertama yaitu IV dibuat secara acak, IV pada langkah-langkah selanjutnya menggunakan hasil dari memrosesan AES terhadap IV langkah sebelumnya. Dan sama seperti langkah sebelumnya, IV baru yang telah diproses dalam AES akan dikombinasikan dengan potongan dari *plaintext* yang berlum terenkripsi dengan operasi *bitwise* XOR. Setelah seluruh *plaintext* terenkripsi menjadi *ciphertext*, tiap blok *ciphertext* hasil operasi XOR digabungkan kembali dan jadilah *ciphertext* akhir dari operasi OFB.

2.8 Python

Python dikembangkan oleh Guido van Rossum di akhir tahun 80-an di *National Research Institute for Mathematics and Computer Science Netherlands*. Python berasal dari beberapa bahasa pemrogramman lain seperti ABC, Modula-3, C, C++, Algol-68, SmallTalk, Unix Shell dan bahasa pemrograman lainnya. Seperti Perl, Python berada dibawah GNU *General Public License* (GPL) [11]. Python adalah salah satu bahasa pemrograman tingkat tinggi yang *interpreted* dan *object-oriented* dengan semantik yang dinamis.

Python merupakan bahasa pemrograman yang sederhana, mudah untuk dipelajari dan karena itu dapat mengurangi biaya perawatan. Sisi utama yang membedakan Python dengan bahasa lain adalah dalam hal aturan penulisan kode program yang mengutamakan indentasi, tipe data, serta tidak memerlukan *semi-colon* (;) di akhir kode tidak seperti kebanyakan Bahasa pemrograman lain. Python juga didukung dengan banyak modul siap pakai dalam bahasa python maupun C/C++ yang membuat python mudah dikembangkan. Saat ini python banyak digunakan untuk pemrograman web, pengembangan *Graphical User Interface* (GUI), perhitungan rumit dalam sains, dan administrasi pada sistem.

2.9 Asterisk

Asterisk [13] adalah implementasi perangkat lunak dari *Telephone Private Branch Exchange* (PBX), diciptakan pada tahun 1999 oleh Mark Spencer dari Digium. Seperti PBX lainnya, Asterisk memungkinkan untuk dipasangkan pesawat telepon dan melakukan panggilan ke satu dengan lainnya, termasuk tersambung ke layanan telepon pribadi dan publik, layanan jaringan telepon umum (PSTN) dan *Voice over Internet Protocol* (VoIP). Nama Asterisk berasal dari * (tanda bintang). Asterisk dirilis dengan model lisensi ganda, menggunakan GNU/GPL sebagai lisensi perangkat lunak bebas (*open source*) dan lisensi perangkat lunak berpemilik untuk mengizinkan pemegang lisensi untuk mendistribusikan komponen sistem proprietari yang tidak perlu dipublikasikan. Awalnya Asterisk dirancang untuk sistem operasi Linux (salah satu distro: AsteriskNOW), namun saat ini Asterisk dapat berjalan pada berbagai sistem operasi lainnya, termasuk NetBSD, OpenBSD, FreeBSD, Mac OS X, dan Solaris.

BAB III
ANALISIS DAN PERANCANGAN PERANGKAT LUNAK

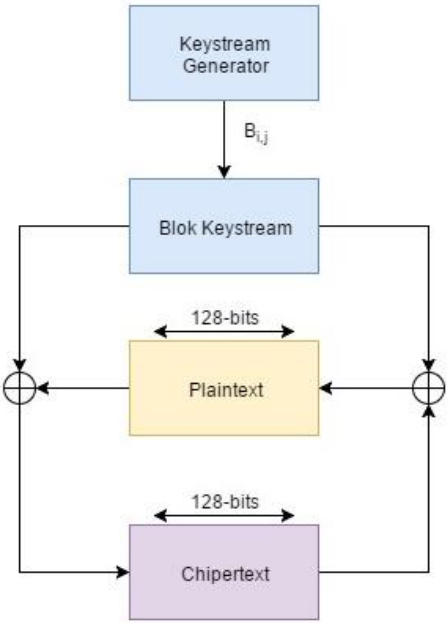
Pada bab ini akan dijelaskan perancangan perangkat lunak yang dibuat. Perancangan akan dibagi menjadi dua tahapan utama, yaitu perancangan program dengan OpenMPI dan peracancangan alur proses utama program. Pada bab ini juga akan dijelaskan mengenai gambaran umum setiap proses utama program dalam diagram alir beserta penjelasannya.

3.1 Penjelasan Umum Algoritma AES-OFB

Pada subbab ini menjelaskan secara umum mengenai algoritma enkripsi *Advanced Encryption Standard* (AES) dengan mode *Integer Counter Mode* (ICM) dan mode *Output Feedback* (OFB).

3.1.1 Algoritma *Integer Counter Mode* (ICM)

Algoritma *Integer Counter Mode* (ICM) atau yang dikenal dengan *Counter* (CTR) atau CM (CTR *mode*) adalah metode enkripsi yang sering digunakan dalam enkripsi data yang *stream* seperti transmisi data multimedia dan juga *Voice over Internet Protocol*. Perbedaan metode ini dengan blok *cipher* terletak pada penggunaan IV yang pada ICM lebih dikenal dengan *keystream* karena metode ini digunakan dalam enkripsi data yang bersifat *streaming*. Maka dari itu, panjang dari k*eystream* yang dibuat pada algoritma ICM akan menyesuaikan panjang dari data yang akan dikirim. Desain umum dari algoritma ICM dapat dilihat pada Gambar 3.1.



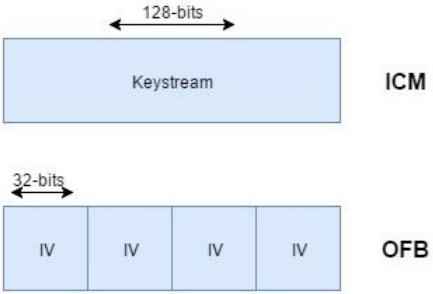
Gambar 3.1 Enkripsi AES ICM Pada SRTP [14]

Pada Gambar 3.1 terdapat *keystream generator* yang berfungsi untuk menghasilkan *keystream* yang akan digunakan pada proses enkripsi. Sebelum *keystream* digunakan pada *plaintext*, *keystream* awal akan diproses terlebih dahulu yang digambarkan pada bagian $B_{i,j}$. Disinilah proses AES dilakukan dengan masukan dari AES adalah IV atau *keystream* awal dan juga kunci AES. Setelah melewati bagian Bi,j maka akan terbentuk *keystream* dengan panjang 128-*bit* yang akan dikombinasikan dengan *plaintext* dengan operasi *bitwise* XOR untuk menghasilkan *ciphertext*.

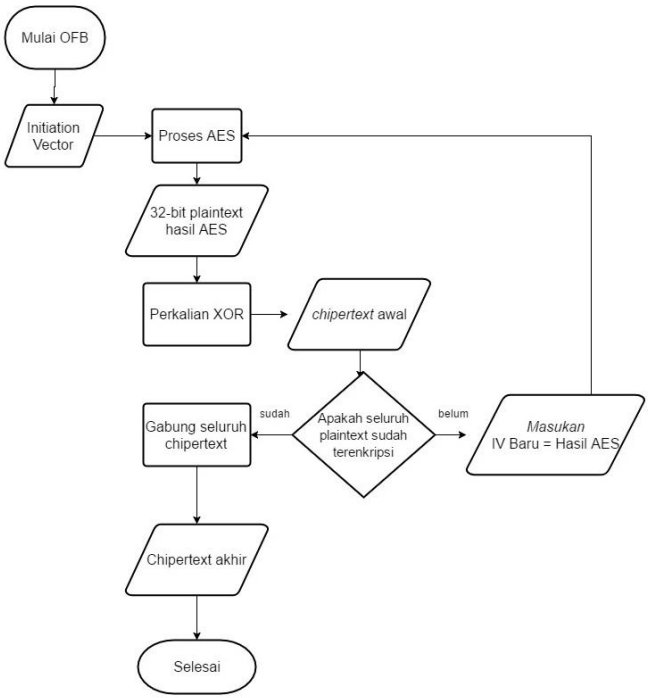
3.1.2 Penyesuaian *Output Feedback* Pada *Integer Counter Mode*

Secara umum algoritma blok *cipher* OFB menyerupai mode ICM. Persamaan dari kedua algortima ini ada pada pemrosesan IV/*keystream* terlebih dahulu yang masuk ke dalam enkripsi AES. Dan pada OFB maupun ICM, keluaran dari proses AES akan dikombinasikan dengan *plaintext* yang akan menghasilkan *ciphertext*. Perbedaan antara OFB dan ICM terletak pada ukuran dari IV pada OFB atau *keystream* pada ICM, dimana IV pada OFB berukuran 32-*bit* sedangkan *keystream* ICM berukuran 128-*bit*. Hal ini dikarenakan pada ICM, *keystream* akan terus dibuat sepanjang *plaintext* dan dienkripsi setiap 128-*bit*, sedangkan untuk OFB yang menggunakan blok *cipher*, hasil keluaran dari blok AES pertama yaitu IV berukuran 32-*bit* yang terenkripsi akan menjadi IV dari blok selanjutnya seperti yang terlihat pada Gambar 3.2.

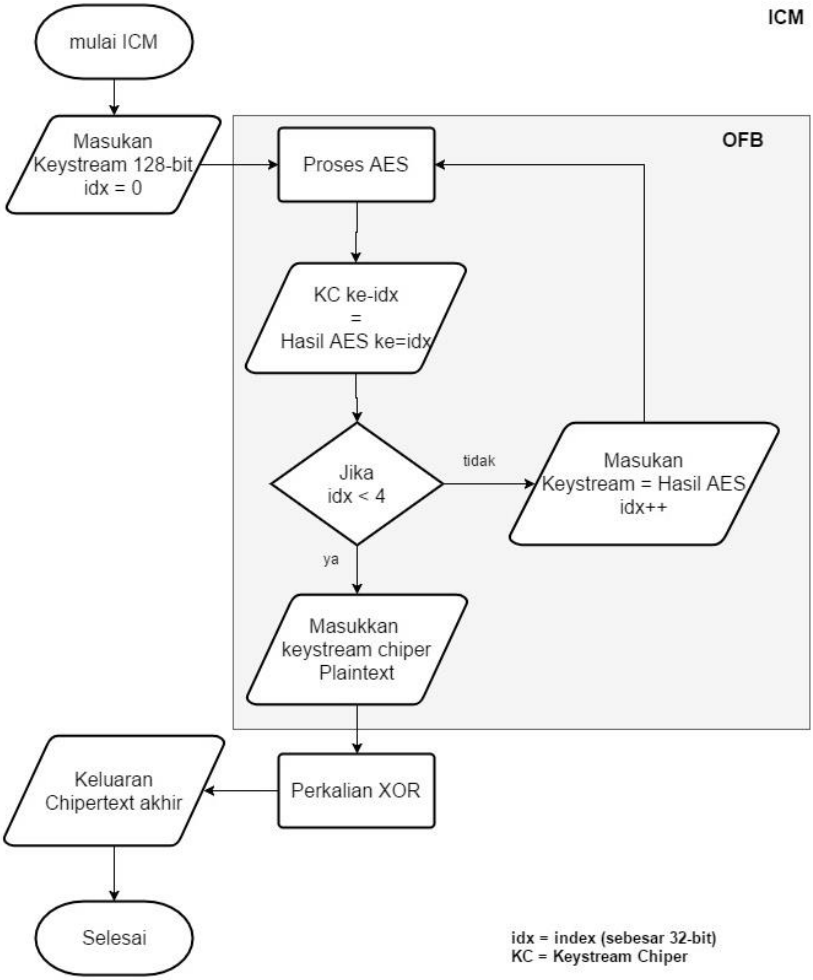
Namun dalam implementasinya, AES ICM memecah operasi *bitwise* XOR antara *keystream cipher* dan *plaintext* menjadi blok-blok yang lebih kecil. Ini dilakukan agar operasi *bitwise* XOR berjalan lebih cepat contohnya 32-*bit* dengan 32-*bit* atau 16-*bit* dengan 16-*bit*. Dari sini kita dapat menyisipkan algoritma OFB pada algoritma ICM yaitu pada proses enkripsi *keystream* yang menggunakan siklus AES. Dengan melakukan beberapa kali perulangan proses AES terhadap *keystream* dari ICM, maka akan didapatkan proses yang sama dengan algoritma OFB. Diagram alir dari algoritma OFB dapat dilihat pada Gambar 3.3. Dan diagram alir OFB pada ICM dapat dilihat terjadi perulangan indeks sebanyak empat kali yang menunjukkan setiap 32-*bit* dari *keystream* akhir melalui proses AES, 32-*bit* pertama melalui 1 siklus AES, 32-*bit* kedua melalui 2 siklus AES dan begitu seterusnya seperti ditunjukkan pada Gambar 3.4.



Gambar 3.2 Perbandingan Keystream pada ICM dan IV pada OFB



Gambar 3.3 Diagram Alir OFB



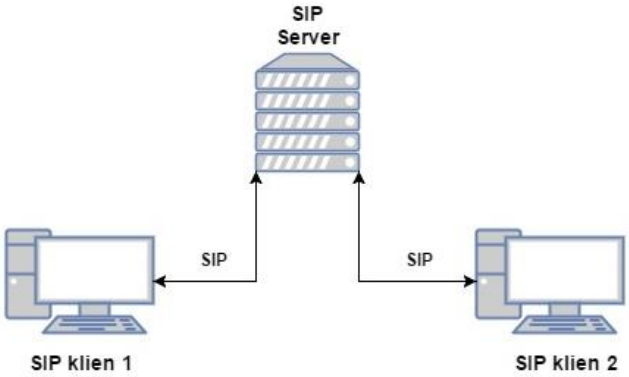
Gambar 3.4 Diagram Alir Proses OFB pada ICM

3.2 Arsitektur Sistem

Sub-bab ini akan membahas mengenai desain dari sistem yang akan diimplementasikan pada tugas akhir.

3.2.1 Desain Umum Sistem

Sistem yang akan dibangun adalah sebuah sistem komunikasi berbasis *Voice over Internet Protocol* (VoIP) yang dapat melakukan panggilan dari satu klien ke klien lainnya melalui sebuah server. Setiap klien akan menggunakan salah satu protokol yang mendukung VoIP yaitu *Session Initiation Protocol* (SIP). Dalam sistem ini dibutuhkan sebuah server sebagai SIP server menggunakan Asterisk yang mendukung VoIP berbasis SIP. Secara umum visualisasi SIP server dapat dilihat pada Gambar 3.5.



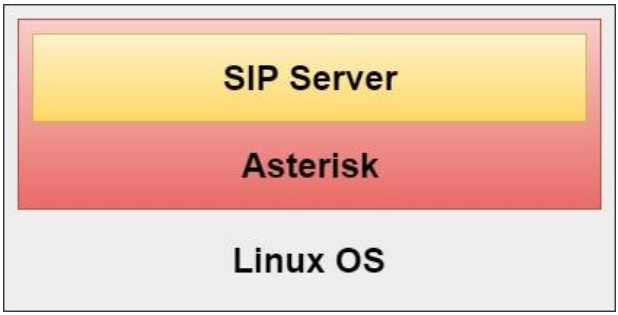
Gambar 3.5 Desain Arsitektur SIP Server dan Klien

Dapat dilihat pada gambar diatas, komponen utama pada system ada dua yaitu SIP server dan SIP klien. Server dibutuhkan sebagai tempat inisialiasi dari akun-akun SIP. Akun SIP dibutuhkan sebagai autentikasi awal untuk tiap SIP klien mengenali satu sama lainnya. Setelah akun SIP dibuat pada SIP server, setiap klien akan mendaftar diri menggunakan akun SIP

yang telah disediakan. Sedangkan SIP klien menggunakan PJSIP yaitu *library* komunikasi multimedia yang bersifat sumber terbuka. *Library* PJSIP ini dibangun dengan bahasa pemrograman C dan mengimplementasikan protokol-protokol standar VoIP salah satunya adalah SIP.

3.2.2 Desain VoIP Server

Server VoIP memiliki peranan penting dalam sistem yang dikembangkan dalam tugas akhir ini. Desain VoIP server dapat dilihat pada Gambar 3.6. Untuk dapat melakukan panggilan dari SIP klien, SIP klien harus terlebih dahulu mendaftar dengan akun-akun SIP yang telah didaftarkan pada konfigurasi server. Apabila saat akan melakukan panggilan akun dari SIP klien tidak ditemukan dalam konfigurasi server atau akun tidak diketahui, maka server akan langsung menolak pendaftaran dari SIP klien. Konsekuensinya SIP klien tidak dapat melakukan panggilan VoIP sampai SIP klien mendaftar dengan akun yang benar/sah.



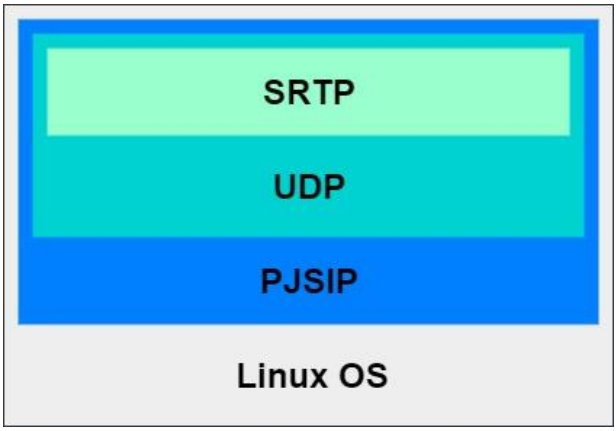
Gambar 3.6 Diagram Arsitektur Server Berbasis Asterisk

Server VoIP akan menggunakan SIP server pada asterisk. Selain mengatur konfigurasi dari akun SIP klien, asterisk juga mengatur beberapa detail dari tiap kliennya. Beberapa konfigurasi pada asterisk untuk tiap kliennya selain id akun antara lain nama akun, pengisian sandi pada akun, *host* (secara standar diisi dengan

dynamic agar server mendeteksi IP *Address* dari klien secara dinamis), *port* (untuk *port* boleh diisi boleh tidak atau opsional), jenis transportasi data (menggunakan transportasi UDP), Codec (*Code-decoder* untuk mengubah data suara yang analog menjadi digital dan sebaliknya), dan enkripsi (secara standar diisi dengan "*no*" atau tanpa enkripsi).

3.2.3 Desain VoIP Klien

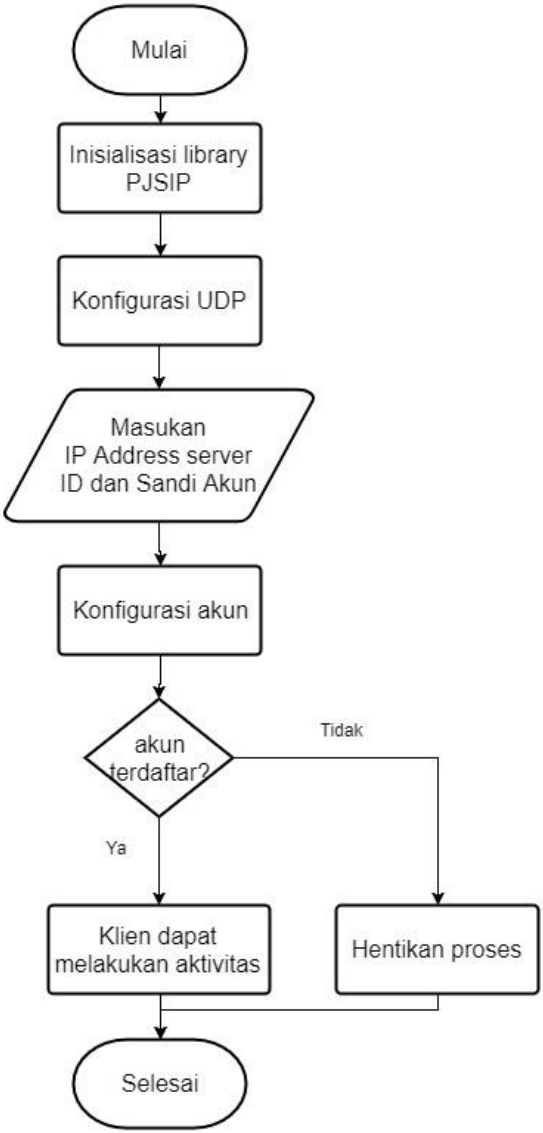
Klien VoIP yang berbasis SIP memiliki fungsi utama untuk melakukan panggilan telepon. Dapat dilihat pada Gambar 3.7, klien akan berjalan pada sistem operasi berbasis Linux dan dibangun dengan *library* PJSIP yang dapat melakukan panggilan telepon standar VoIP seperti SIP. Selain melakukan panggilan, klien harus mendaftarkan akun ke SIP server asterisk. Klien juga harus mendukung *Secure Real-Time Transport Protocol* (SRTP) yang berjalan diatas *User Datagram Protocol* (UDP) sebagai protokol pengiriman data suara dari tiap klien.



Gambar 3.7 Diagram Arsitektur VoIP Klien

Pada *library* PJSIP terdapat *Application Programming Interface* (API) dalam bahasa pemrograman Python yang bernama PJSUA. PJSUA ini merupakan API tingkat tinggi untuk membangun klien multimedia berbasis SIP dari PJSIP. PJSUA sekaligus mendukung seluruh pensinyalan dan fungsionalitas media yang mudah di implementasikan. PJSUA juga mendukung manajemen akun pada API-nya seperti pengisian id akun, serta jenis transportasi media yang dipergunakan akun. Untuk jalannya klien sendiri dimulai dengan inisialisasi dari *library* terlebih dahulu. Apabila library telah ter-*install* sempurna kemudian klien akan mengatur media transportasi UDP. Disini klien akan secara otomatis mendeteksi IP *Address*-nya sendiri serta *port* yang dapat dipergunakan oleh klien dalam komunikasi UDP nantinya. Setelah itu klien akan mendaftar ke SIP server dengan menggunakan akun yang sebelumnya sudah diatur oleh SIP server. Jika pendaftaran akun sukses maka klien siap untuk melakukan aktivitas selanjutnya. Namun jika pendaftaran gagal karena akun tidak terdaftar atau IP *Address* dari server tidak diketahui, maka *library* akan langsung berhenti berjalan. Jalan konfigurasi dari klien dapat dilihat pada Gambar 3.8.

Tepat setelah pendaftaran selesai, klien akan melakukan konfigurasi akun lebih jauh seperti memberi id pada akun, serta mengaktifkan SRTP sehingga nanti SRTP akan berjalan diatas UDP saat panggilan berlangsung. Dan tidak kalah pentingnya yaitu menyiapkan *callback* dari akun klien sehingga akun dari klien dapat menerima telepon dari SIP klien lainnya. Setelah selesai melakukan konfigurasi tambahan pada akun, selanjutnya klien dapat melakukan aktivitas VoIP seperti melakukan panggilan, menerima panggilan, atau keluar. Jalan aktivitas klien dapat dilihat pada Gambar 3.9.



Gambar 3.8 Diagram Alir Konfigurasi Akun SIP Klien

Gambar 3.9 Diagram Alir Aktivitas SIP Klien

[Halaman ini sengaja dikosongkan]

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Sebelum penjelasan implementasi akan ditunjukkan terlebih dahulu lingkungan untuk melakukan implementasi.

4.1 Lingkungan Implementasi

Lingkungan implementasi dan pengembangan dilakukan menggunakan komputer dengan spesifikasi Intel® Core™ i7-4500U CPU @ 1.80GHz (4 CPUs) dengan *memory* sebesar 4GB. Perangkat lunak uang digunakan untuk melakukan proses implementasi adalah:

- Sistem operasi Linux Ubuntu 14.04 LTS.
- Editor teks Atom.
- Python 2.7 sebagai bahasa pemrograman utama klien
- Asterisk 14.4.0 sebagai VoIP server.
- Libpri sebagai *dependency* dari Asterisk untuk beberapa protokol komunikasi.
- DAHDI (Digium / Asterisk Hardware Device Interface) sebagai *dependency* dari Asterisk untuk mendukung penggunaan *hardware.*
- PJSIP 2.6 sebagai *library* dari VoIP klien.
- Libsrtp sebagai *library* dari protokol media SRTP.
- PJSUA sebagai API dari PJSIP dalam bahasa pemrograman Python.

4.2 Rincian Implementasi VoIP Server

VoIP server digunakan sebagai tempat konfigurasi akun-akun SIP klien, menerima *request* dari SIP klien dan mengirim respon ke SIP klien sesuai kebutuhan klien. Dalam pengerjaan tugas akhir ini, VoIP server yang digunakan adalah asterisk. Pada

sub bab ini akan dijelaskan secara rinci mengenai instalasi Asterisk dan konfigurasi akun SIP klien.

4.2.1 Instalasi Server Berbasis Asterisk

Untuk melakukan instalasi Asterisk pada komputer server, sebaiknya terlebih dahulu mengunduh paket Asterisk, Dahdi, dan Libpri, serta Libsrtp karena modul SRTP pada Asterisk tidak dapat berjalan tanpa instalasi *library* dari SRTP terlebih dahulu. Untuk instalasi ini dibutuhkan akses *super user* atau *root*. Langkah-langkah instalasi server antara lain:

1. Siapkan semua *dependency* dasar yang dibutuhkan untuk membangun *library* seperti GNU make, GNU binutils dan GNU gcc. Jika telah disiapkan pastikan semua telah diperbarui dengan *apt-get upgrade* dan *apt-get update*.
2. Unduh paket instalasi dari masing-masing sumber melalui *web browser* atau melakukan *clone* dari masing-masing *repository* github Asterisk dan SRTP. *git clone https://github.com/asterisk/libpri.git* untuk Libpri, *git clone https://github.com/asterisk/dahdi-linux.git* dan *git clone https://github.com/asterisk/dahdi-tools.git* untuk Dahdi, *git clone https://github.com/asterisk/asterisk.git* untuk Asterisk dan *git clone https://github.com/cisco/libsrtp.git* untuk library SRTP.
3. Untuk instalasi libpri, masuk ke dalam direktori libpri dengan *cd libpri* lalu lakukan *make && make install*. Setelah proses selesai keluarlah ke direktori awal.
4. Selanjutnya untuk proses instalasi Dahdi terdapat dua kali instalasi yaitu pada Dahdi Linux dan juga Dahdi Tools.
5. Untuk instalasi Dahdi Linux, masuk ke dalam direktori dengan *cd dahdi-linux* lalu lakukan *make && make install*. Setelah proses selesai keluarlah ke direktori awal.
6. Untuk instalasi Dahdi Tools, masuk ke dalam direktori dengan *cd dahdi-tools* lalu lakukan *autoreconf -i*. selanjutnya lakukan *./configure* dilanjutkan dengan *make*

&& make install. Jika dibutuhkan instalasi tambahan maka jalankan *make install-config*. Setelah proses selesai keluarlah ke direktori awal.

7. Untuk instalasi Libsrtp, masuk ke dalam direktori libsrtp dengan cd libsrtp lalu lakukan *./configure CFLAGS=-fPIC --prefix=/usr* agar libsrtp menjadi *shared library* atau semua program lain dapat menggunakan *library* ini. Dilanjutkan dengan *make && make runtest && make install*. Setelah proses selesai keluarlah ke direktori awal.
8. Yang terakhir adalah instalasi Asterisk. Masuk ke dalam direktori asterisk dengan *cd asterisk* lalu lakukan *./configure* dilanjutkan dengan *make menuselect && make*. Pada saat *make menuselect* pastikan modul SRTP terpilih yang ditandakan dengan notasi ✓. Dilanjutkan dengan *make install && make samples*.

Apabila langkah-langkah instalasi sudah selesai dilakukan, maka coba jalankan server Asterisk dengan *asterisk –vvvc* (inisialisasi Asterisk dengan mode "*very very verbose*") dan akan muncul konsol dari Asterisk seperti yang ditunjukkan pada Gambar 4.1.

```
dwi@dwi-S451LB:~$ sudo asterisk -rvvvc
[sudo] password for dwi:
Asterisk GIT-master-62386dd, Copyright (C) 1999 - 2016, Digium, Inc. and others.
Created by Mark Spencer <markster@digium.com>
Asterisk comes with ABSOLUTELY NO WARRANTY; type 'core show warranty' for detail
s.
This is free software, with components licensed under the GNU General Public
License version 2 and other licenses; you are welcome to redistribute it under
certain conditions. Type 'core show license' for details.
=========================================================================
Connected to Asterisk GIT-master-62386dd currently running on dwi-S451LB (pid =
1098)
dwi-S451LB*CLI>
```

Gambar 4.1 Konsol Asterisk

4.2.2 Konfigurasi Akun SIP Klien

Setelah melakukan instalasi Asterisk pada server dan Asterisk berjalan dengan baik, maka selanjutnya lakukan

konfigurasi akun SIP klien. Konfigurasi ini dilakukan pada dua berkas yaitu *sip.conf* dan *extensions.conf*. Berkas *sip.conf* sebagai konfigurasi akun SIP dan berkas *extensions.conf* agar akun dapat dihubungi dengan menggunakan dialplan. Untuk konfigurasi ini dibutuhkan akses *super user* atau *root*. Langkah-langkah konfigurasi akun SIP antara lain:

1. Buka terminal Ubuntu lalu masuk ke dalam direktori Asterisk yang berada di direktori */etc* dengan *cd /etc/asterisk*.
2. Pada direktori Asterisk, buka dan sunting berkas *sip.conf* dengan text editor terminal *vim sip.conf*. Isi konfigurasi pada berkas ini meliputi id akun, *host*, sandi/*secret*, Codec yang digunakan, transportasi media, dan mode enkripsi.
3. Setelah konfigurasi akun pada berkas *sip.conf* selesai, simpan perubahan.
4. Selanjutnya buka dan sunting berkas extensions.conf dengan text editor terminal *vim extensions.conf*. Isi konfigurasi pada berkas ini meliputi kemampuan untuk *Answer*, *Hang Up*, *Dial*, dan beberapa konfigurasi SRTP.
5. Setelah konfigurasi akun pada berkas *extensions.conf* selesai, simpan perubahan.
6. Jika tidak terbiasa menggunakan editor teks pada terminal Ubuntu, konfigurasi *sip.conf* dan *extensions.conf* dapat menggunakan editor teks seperti Atom namun harus berjalankan dengan akses *super user* atau *root*.
7. Selanjutnya jalankan Asterisk dengan *asterisk -vvvc*, jika sudah pernah menjalankan Asterisk dapat menggunakan konsol *remote* Asterisk dengan *asterisk -rvvv*. Pada konsol lakukan *sip reload* untuk eksekusi ulang berkas *sip.conf* dan *dialplan reload* untuk eksekusi ulang berkas *extensions.conf*.

4.3 Rincian Implementasi VoIP Klien

VoIP klien berfungsi untuk melakukan panggilan VoIP melalui server. *Library* yang digunakan adalah PJSIP yang mendukung implementasi klien sesuai kebutuhan tugas akhir. Klien akan dibangun dengan PJSUA yaitu API dari PJSIP yang dibangun dengan bahasa pemrograman Python. Pada sub bab ini akan dijelaskan secara rinci mengenai instalasi PJSIP, PJSUA dan implementasi SIP klien.

4.3.1 Instalasi PJSIP

Untuk melakukan instalasi PJSIP pada komputer klien, sebaiknya terlebih dahulu mengunduh paket PJSIP. Untuk instalasi ini dibutuhkan akses *super user* atau *root*. Langkah-langkah instalasi antara lain:

1. Siapkan semua *dependency* dasar yang dibutuhkan untuk membangun *library* seperti GNU make, GNU binutils dan GNU gcc. Jika telah disiapkan pastikan semua telah diperbarui dengan *apt-get upgrade* dan *apt-get update*.
2. Setelah mengunduh paket instalasi PJSIP, ekstrak paket dengan *tar xvjf pjproject-2.6.tar.bz2*.
3. Pindahkan seluruh berkas instalasi pada folder *pjproject-2.6* ke direktori */etc/usr/pjproject.*
4. Selanjutnya mulai jalankan instalasi, pertama-tama masuk ke direktori berkas instalasi dengan *cd /etc/usr/pjproject* lalu lakukan *export CFLAGS=”$CFLAGS -fPIC” && ./configure*. Penggunaan -fPIC dimaksudkan agar PJSIP menjadi *shared library*. Setelah itu lakukan *make dep && make && make install*.
5. Selanjutnya untuk instalasi modul Python untuk PJSUA, masuk ke dalam direktori berkas instalasi dengan *cd*

pjsip-apps/src/python lalu jalankan *python ./setup.py install*.

Apabila langkah-langkah instalasi sudah selesai dilakukan, maka coba jalankan Python pada terminal Ubuntu dan *import pjsua*. Jika tidak muncul pesan error maka instalasi PJSIP dan API PJSUA berhasil.

4.3.2 Implementasi SIP Klien

Setelah menyelesaikan langkah-langkah instalasi PJSIP dan PJSUA, maka SIP klien dapat diimplementasikan. Sesuai namanya, SIP klien dibangun berbasis protokol komunikasi dan pensinyalan SIP dan UDP sebagai protokol media. Implementasi dari klien dibangun dalam bahasa pemrograman Python dengan memanfaatkan API dari PJSIP yaitu PJSUA. Klien juga dapat melakukan panggilan aman/*secure* yang memanfaatkan SRTP untuk mengenkripsi data-data suara dari klien.

Terdapat dua bagian utama dalam implementasi dari SIP klien yaitu bagian inisialisasi dan bagian melakukan aktivitas. Untuk bagian inisialisasi berisi semua inisialisasi dari *library* PJSUA, mengatur konfigurasi transportasi media pada klien, mengatur konfigurasi akun SIP klien, dan menghentikan/*destroy library* PJSUA saat terjadi *error*.

Berikut penjelasan dari Pseudocode 1:

1. Implementasi SIP Klien dimulai dengan inisialisasi dari *library* PJSUA pada baris 1.
2. Pada baris 2 atur log_level sama dengan 3 artinya tingkat verbositas dari akun SIP klien adalah normal (skala 1 sampai 6).
3. Pada baris 3 yaitu inisialisasi variabel current_call = 0
4. Pada baris 4 dan 5 proses inisialisasi media transportasi, dengan begitu program akan mengatur sendiri IP *Address* dan *Port* yang digunakan. *Transport* yang digunakan adalah UDP.
5. Pada baris 7 mulai jalankan *library* PJSUA.

6. Baris 8 sampai 10 masukkan IP *address* dari server, id akun SIP yang akan didaftarkan, dan kata sandinya.
7. Selanjutnya pada baris 11 lakukan inisialisasi akun SIP ke server dengan masukan sebelumnya.
8. Baris 13 sampai dengan 19, apabila inisialisasi sukses dijalankan, maka proses dilanjutkan dengan inisialisasi konfigurasi akun pada SIP klien dan membuat *callback* dari akun.
9. Pada bagian inisialiasi juga mengatur konfigurasi dari SRTP klien. Jika ingin mengharuskan klien menggunakan SRTP, maka atur *acc_conf_srtp* dengan 2. Jika hanya opsional saja untuk penggunaan SRTP, maka atur *acc_conf_srtp* dengan 0 (tanpa SRTP) atau 1 (Opsional). Untuk variabel *acc_conf_signal_srtp* di atur dengan 0 karena klien menggunakan SRTP tanpa bantuan pensinyalan yang dienkripsi yaitu *Transport Layer Security* (TLS).
10. Baris 21 sampai 24, apabila inisialisasi gagal dilakukan, maka variabel-variabel akan dijadikan NULL lalu *library* dihentikan/*destroy*.

```
 1  initialize pjsua
 2  set log_level = 3
 3  set current_call = 0
 4  set transport
 5  transport = UDP
 6
 7  start pjsua
 8  get server_ip_address
 9  get user_account
10  get password
11  set acc_conf
12
13  if (acc_conf is success)
14      set acc_conf_id = user_account
15      set acc_conf_uri = server_ip_address+transport_port
16      set acc_conf_srtp = 2
```

```
17      set acc_conf_signal_srtp = 0
18      set acc_callback with acc_conf
19      execute acc_callback
20
21  else
22      set transport = none
23      set acc_conf = none
24      destroy pjsua
```

Pseudocode 1 Inisialisasi SIP Klien

Selanjutnya pada bagian aktivitas klien, SIP klien akan masuk ke dalam sebuah perulangan *while* yang dapat memilih beberapa aktivitas yang dapat di pilih dengan memasukkan karakter yang merepresentasikan aktivitas tersebut. Adapun aktivitas yang dapat dijalankan SIP klien antara lain melakukan panggilan (*call*) ke akun lainnya dengan memasukkan alamat klien dengan benar, menerima panggilan (*answer*) dari akun klien lain, menutup panggilan (*hangup*) yang sedang berlangsung, dan keluar dari menu (*quit*) jika sudah selesai dan tidak akan melakukan aktivitas lainnya. Akun dari SIP klien yang dipakai pada saat mendaftarkan akun tidak dapat diganti setelah mendaftar karena tidak disediakan fitur untuk mengganti akun pada aktivitas. Satu-satunya cara untuk mengganti akun klien adalah dengan keluar terlebih dahulu lalu mengeksekusi ulang SIP klien dari awal.

Berikut penjelasan dari Pseudocode 2:

1. Pada baris 1 yaitu saat memasuki bagian aktivitas, SIP klien akan masuk ke dalam sebuah perulangan dengan kodisi yang selalu benar (*True*).
2. Pada baris 2, setelah itu masukkan aktivitas dalam variable *input.*
3. Lakukan pengecekan terdahap masukan dalam variabel *input*.
4. Pada baris 3 sampai 9 yaitu jika variabel *input* berisi *call* maka lakukan pengecekan kembali pada variabel *current_call*. Jika *current_call* tidak tidak kosong maka lakukan *print* dan kembali ke perulangan awal dan masukkan

kembali *input*. Jika *current_call* kosong maka masukkan alamat klien yang akan dipanggil lalu lakukan panggilan VoIP dengan mengeksekusi fungsi *make_call*.

5. Pada baris 10 sampai 15 yaitu jika variabel *input* berisi *hangup* maka lakukan pengecekan kembali pada variabel *current_call*. Jika *current_call* kosong maka lakukan *print* dan kembali ke perulangan awal dan masukkan kembali *input*. Jika *current_call* tidak kosong maka eksekusi fungsi *hangup* untuk menutup panggilan masuk.
6. Pada baris 16 sampai dengan 21 yaitu jika variabel *input* berisi *answer* maka lakukan pengecekan kembali pada variabel *current_call*. Jika *current_call* kosong maka lakukan *print* dan kembali ke perulangan awal dan masukkan kembali *input*. Jika *current_call* tidak kosong maka eksekusi fungsi *answer* untuk menjawab panggilan masuk.
7. Dan pada baris 22 dan 23 yaitu jika variabel input berisi *quit* maka lakukan *break* dan perulangan akan berhenti dan SIP klien akan dimatikan.

```
1   while True
2       get input
3       if (input = call)
4           if (current_call is not empty)
5               print "already have a call"
6               continue
7           get destination_uri
8           set current_call = destination uri
9           execute current_call call
10      else if (input = hangup)
11          if (current_call is empty)
12              print "no call"
13              continue
14          else
15              execute current_call hangup
16      else if (input = answer)
17          if (current_call is empty)
18              print "no call"
```

```
19              continue
20          else
21              execute current_call answer
22      else if (input = quit)
23          break
```

Pseudocode 2 Aktivitas SIP Klien

4.4 Implementasi AES OFB pada Libsrtp

Libsrtp adalah *library* yang mendukung protokol media SRTP yang berjalan pada UDP. SRTP memungkinkan server dan klien melakukan enkripsi pada data-data suara. Libsrtp sendiri menggunakan AES ICM sebagai mode enkripsi utamanya. Untuk itu maka dibutuhkan implementasi mode OFB pada AES ICM. Pada umumnya mode OFB dan mode ICM memiliki banyak kesamaan khususnya pada pemrosesan *plaintext* menjadi *ciphertext* yang dilakukan di akhir. Perbedaannya terletak pada pembentukan *keystream cipher* dengan ICM membentuk *keystream cipher* sepanjang *bit* dari *plaintext* sedangkan OFB membentuk *keystream* dengan IV hanya 32-*bit* yang melewati beberapa blok AES.

```
1   set count = 0
2   get keystream
3   get buffer
4   set iv_aes = keystream
5
6   while (count < 4)
7       execute aes_encrypt(iv_aes)
8       set keystream[count] = iv_aes[count]
9       set count++
10
11  execute buffer XOR keystream
```

Pseudocode 3 Implementasi OFB pada Libsrtp

Berikut penjelasan dari Pseudocode 3:

1. Saat memasuki proses enkripsi *keystream*, inisialiasi variabel yaitu *count* sebagai iterasi perulangan operasi AES.
2. Dapatkan *keystream* awal dan data *buffer* atau data *plaintext*.
3. Masukkan isi dari keystream ke dalam variabel baru yaitu *iv_aes*.
4. Lakukan perulangan sebanyak empat kali karena *keystream* awal berukuran 128-*bit* sedangkan dalam OFB *keystream* nanti akan dibagi menjadi data berukuran 32-*bit* sehingga diperlukan empat kali perulangan. Yang masuk ke dalam enkripsi AES adalah *iv_aes*.
5. Setelah enkripsi AES dilakukan, masukkan *keystream* dengan *iv_aes* sesuai dengan 32-*bit* indeks dari variabel *count*. Lalu lakukan proses *increment* (tambahkan satu) pada *count*.
6. Setelah seluruh proses perulangan selesai, maka *keystream cipher* akan dikombinasikan dengan *plaintext* menjadi *ciphertext*.

[Halaman ini sengaja dikosongkan]

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan dijelaskan uji coba yang dilakukan pada aplikasi yang telah dikerjakan serta analisa dari uji coba yang telah dilakukan. Pembahasan pengujian meliputi lingkungan uji coba, skenario uji coba yang meliputi uji kebenaran dan uji kinerja serta analisa setiap pengujian.

5.1 Lingkungan Uji Coba

Lingkungan uji coba menjelaskan lingkungan yang digunakan untuk menguji program SIP klien dalam melakukan panggilan VoIP. Lingkungan uji coba untuk server VoIP meliputi perangkat keras dan perangkat lunak yang dijelaskan pada Tabel 5.1.

Tabel 5.1 Spesifikasi Perangkat Keras dan Perangkat Lunak Komputer Server

No	IP Komputer	Prosesor	RAM	Tipe Sistem Operasi
1	10.151.43.252	Intel® Core™ i7-4500U CPU @ 1.80GHz (4 CPUs)	3.8 GB	Ubuntu 64-bit

Lingkungan uji coba untuk klien VoIP meliputi perangkat keras dan perangkat lunak yang dijelaskan pada Tabel 5.2.

Tabel 5.2 Spesifikasi Perangkat Keras dan Perangkat Lunak Komputer Klien

No	IP Komputer	Prosesor	RAM	Tipe Sistem Operasi
1	10.151.43.252	Intel® Core™ i7-4500U CPU @ 1.80GHz (4 CPUs)	3.8 GB	Ubuntu 64-bit
2	10.151.36.41	Intel® Core™ 2 Duo CPU E4600 @ 1.80GHz (4 CPUs)	1.9 GB	Ubuntu 64-bit

5.2 *Dataset* Uji Coba

Pada tugas akhir ini, penulis menggunakan sebuah *file* audio yang dikirim oleh salah satu klien dengan menggunakan VoIP. *File* audio yang digunakan dalam uji coba memiliki spesifikasi sebagai berikut:

- Format audio : wave (.wav)
- Nama file : virgoun.wav
- Ukuran : 22.9 Mb
- Bitrate : 705 kbps
- Durasi : 4 menit 33 detik

5.3 Skenario dan Evaluasi Pengujian

Uji coba ini dilakukan untuk menguji apakah fungsionalitas program telah diimplementasikan dengan benar dan berjalan sebagaimana mestinya. Uji coba akan didasarkan pada beberapa skenario untuk menguji kesesuaian dan kinerja aplikasi.

Skenario pengujian terdiri dari tiga skenario pengujian yaitu:

1. Skenario efek penggunaan *codec* GSM pada panggilan VoIP tanpa enkripsi, enkripsi mode ICM 128, enkripsi mode ICM

256, enkripsi mode OFB 128, dan enkripsi mode OFB 256. Variasi waktu yang digunakan selama uji coba panggilan VoIP adalah 50 detik, 100 detik, 150 detik, 200 detik, dan 250 detik. Skenario uji coba 1 dibagi menjadi empat bagian yaitu:

a. Pengaruh skenario uji coba 1 terhadap besar *traffic* data RTP
b. Pengaruh skenario uji coba 1 terhadap penggunaan RAM pada klien VoIP
c. Pengaruh skenario uji coba 1 terhadap *jitter* maksimal
d. Pengaruh skenario uji coba 1 terhadap rata-rata *jitter*

2. Skenario efek penggunaan *codec* G.722 pada panggilan VoIP tanpa enkripsi, enkripsi mode ICM 128, enkripsi mode ICM 256, enkripsi mode OFB 128, dan enkripsi mode OFB 256. Variasi waktu yang digunakan selama uji coba panggilan VoIP adalah 50 detik, 100 detik, 150 detik, 200 detik, dan 250 detik. Skenario uji coba 2 dibagi menjadi empat bagian yaitu:
   a. Pengaruh skenario uji coba 2 terhadap besar *traffic* data RTP
   b. Pengaruh skenario uji coba 2 terhadap penggunaan RAM pada klien VoIP
   c. Pengaruh skenario uji coba 2 terhadap *jitter* maksimal
   d. Pengaruh skenario uji coba 2 terhadap rata-rata *jitter*
3. Skenario efek penggunaan *codec* G.711 pada panggilan VoIP tanpa enkripsi, enkripsi mode ICM 128, enkripsi mode ICM 256, enkripsi mode OFB 128, dan enkripsi mode OFB 256. Variasi waktu yang digunakan selama uji coba panggilan VoIP adalah 50 detik, 100 detik, 150 detik, 200 detik, dan 250 detik. Skenario uji coba 3 dibagi menjadi empat bagian yaitu:
   a. Pengaruh skenario uji coba 3 terhadap besar *traffic* data RTP
   b. Pengaruh skenario uji coba 3 terhadap penggunaan RAM pada klien VoIP

c. Pengaruh skenario uji coba 3 terhadap *jitter* maksimal
d. Pengaruh skenario uji coba 3 terhadap rata-rata *jitter*

5.3.1 Skenario Uji Coba 1

Skenario uji coba 1 adalah pengujian panggilan VoIP dengan *codec* GSM pada klien yang mengimplementasikan mode tanpa enkripsi, enkripsi mode ICM 128, enkripsi mode ICM 256, enkripsi mode OFB 128, dan enkripsi mode OFB 256. Variasi waktu yang digunakan pada uji coba 1 adalah 50 detik, 100 detik, 150 detik, 200 detik, dan 250 detik. Data yang dicatat dan dievaluasi pada uji coba 1 adalah *traffic* data RTP yang memuat data suara dari klien, penggunaan RAM maksimal, *jitter* maksimal dan rata-rata *jitter*.

5.3.1.1 Skenario Uji Coba 1a

Skenario uji coba 1a adalah pengujian panggilan VoIP dengan *codec* GSM pada klien yang mengimplementasikan mode tanpa enkripsi, enkripsi mode ICM 128, enkripsi mode ICM 256, enkripsi mode OFB 128, dan enkripsi mode OFB 256. Variasi waktu yang digunakan pada uji coba 1a adalah 50 detik, 100 detik, 150 detik, 200 detik, dan 250 detik. Dalam uji coba 1a, data yang dicatat adalah *traffic* dari data RTP yang memuat data suara dari klien. Nilai tertinggi di tiap variasi waktu diberi latar belakang warna merah sedangkan nilai terendah berwarna hijau dengan tulisan berwarna putih.

5.3.1.2 Evaluasi Uji Coba 1a

Setelah dilakukan uji coba seperti pada skenario 1a, didapatkan hasil berupa total *traffic* data RTP yang ditunjukkan oleh Gambar 5.1 dan hasil secara rinci pada Tabel 5.3. Pada Gambar 5.1 dapat diamati bahwa total *traffic* data RTP selalu mengingkat atau berbanding lurus dengan penambahan durasi

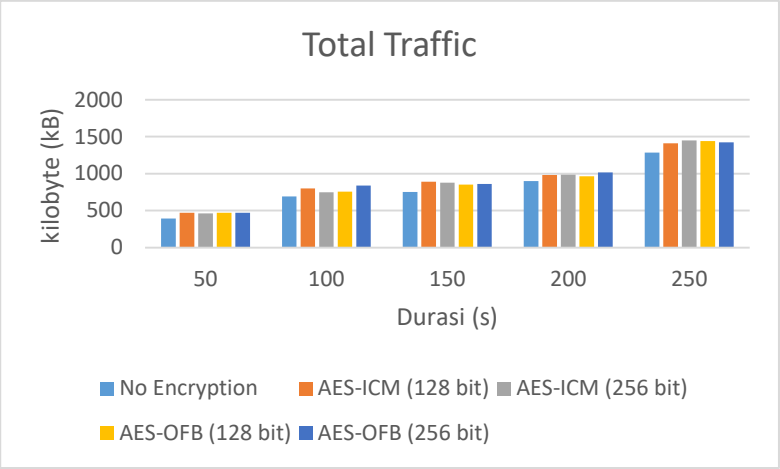
panggilan VoIP. Secara keseluruhan, panggilan VoIP tanpa menggunakan mode enkripsi mendapatkan total traffic RTP yang lebih kecil daripada menggunakan mode enkripsi. Ini disebabkan ukuran sebuah paket RTP dengan *codec* GSM pada mode tanpa enkripsi 87 *byte*, sedangkan dengan enkripsi 97 *byte*.

Untuk awal panggilan VoIP, terlihat AES-OFB 256 mengirim paket RTP terbesar yaitu 20% lebih banyak dari total paket RTP mode tanpa enkripsi. Selanjutnya pada durasi 100 detik mode AES-OFB 256 kembali mengirim paket RTP terbesar yaitu 21%. Pada durasi 150 detik giliran mode AES-ICM 128 mengirim paket terbesar yaitu 18%. Pada durasi 200 detik, AES-OFB 256 mengirim data RTP lebih banyak 13% sedangkan durasi 250 detik AES-ICM 256 mengirim data RTP lebih banyak 12% dari mode tanpa enkripsi. Ini menunjukkan mode enkripsi pada panggilan VoIP menggunakan *codec* GSM memroses banyak paket RTP yang dapat memenuhi *traffic* pada jaringan.

AES-OFB 128 mengirim lebih sedikit paket SRTP dari AES-ICM 128 pada durasi awal hingga durasi 200 detik, selanjutnya pada durasi 250 AES-OFB 128 mengirim lebih banyak yang menunjukkan hanya di akhir panggilan AES-OFB 128 lebih membebani jaringan daripada AES-ICM 128. AES-OFB 256 mengirim lebih banyak paket SRTP dari AES-ICM 256 pada durasi awal hingga durasi 200 yang menunjukkan di awal dan pertengahan panggilan AES-OFB 256 lebih membebani jaringan daripada AES-ICM 256. Untuk keseluruhan mode enkripsi memiliki nilai *throughput* lebih tinggi dari *bandwidth* GSM (1.625 kBps) sehingga paket dapat melewati jaringan dengan baik.

Jika dilihat Tabel 5.4, tren dari *throughput* cenderung terus menurun pada seluruh mode enkripsi dari durasi 50 detik hingga durasi 200 detik dan kembali naik pada durasi 250 detik. Ini disebabkan terjadinya penumpukan paket RTP pada jaringan yang membuat klien mengurangi pengiriman paket RTP sehingga dapat menyebabkan kualitas suara menurun. Artinya setelah detik 200 kualitas suara dari panggilan berangsur membaik dengan nilai *throughput* tertinggi adalah AES-ICM 256 dengan 5.795 kBps

dengan GSM mempengaruhi *bandwidth* sebesar 28% sedangkan untuk mode AES-OFB 128 lebih rendah 0.5% dan AES-OFB 256 lebih rendah 1.6% dengan GSM mempengaruhi *bandwidth* sebesar 28% dan 28.5%.



Gambar 5.1 Grafik Total Traffic RTP pada Codec GSM

Tabel 5.3 Hasil Uji Coba Panggilan VoIP terhadap Total Traffic dengan Codec GSM (kB)

Enkripsi	Durasi Panggilan				
	50s	100s	150s	200s	250s
No Encryption	**393.301**	**692.364**	**752.958**	**899.719**	**1286.529**
AES-ICM (128 bit)	470.943	801.481	**890.733**	982.655	1411.483
AES-ICM (256 bit)	463.440	747.790	877.187	987.259	**1448.805**
AES-OFB (128 bit)	468.915	758.400	850.114	963.521	1441.359
AES-OFB (256 bit)	**472.591**	**839.296**	860.553	**1018.443**	1424.195

Tabel 5.4 Perhitungan Throughput pada Codec GSM (kBps)

Enkripsi	Durasi Panggilan				
	50s	100s	150s	200s	250s
No Encryption	**7.866**	**6.924**	**5.020**	**4.499**	**5.146**
AES-ICM (128 bit)	9.419	8.015	**5.938**	4.913	5.646
AES-ICM (256 bit)	9.269	7.478	5.848	4.936	**5.795**
AES-OFB (128 bit)	9.378	7.584	5.667	4.818	5.765
AES-OFB (256 bit)	**9.452**	**8.393**	5.737	**5.092**	5.697

5.3.1.3 Skenario Uji Coba 1b

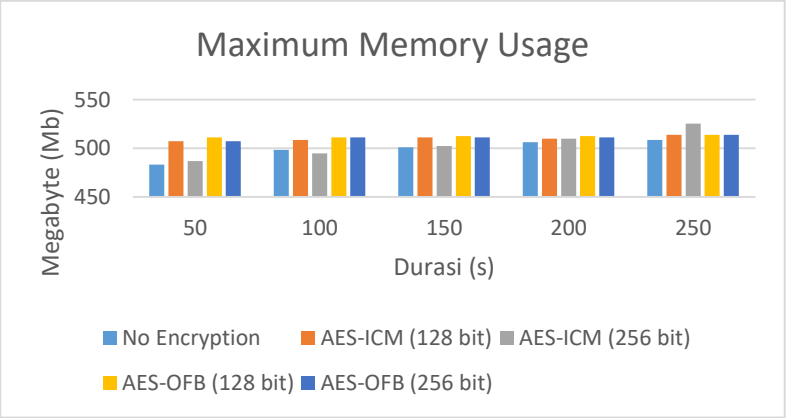
Skenario uji coba 1b adalah pengujian panggilan VoIP dengan *codec* GSM pada klien yang mengimplementasikan mode tanpa enkripsi, enkripsi mode ICM 128, enkripsi mode ICM 256, enkripsi mode OFB 128, dan enkripsi mode OFB 256. Variasi waktu yang digunakan pada uji coba 1b adalah 50 detik, 100 detik, 150 detik, 200 detik, dan 250 detik. Dalam uji coba 1b, data yang dicatat adalah penggunaan RAM pada saat panggilan VoIP berlangsung. Nilai tertinggi di tiap variasi waktu diberi latar belakang warna merah sedangkan nilai terendah berwarna hijau dengan tulisan berwarna putih.

5.3.1.4 Evaluasi Uji Coba 1b

Setelah dilakukan uji coba seperti pada skenario 1b, didapatkan hasil berupa penggunaan maksimal RAM yang ditunjukkan oleh Gambar 5.2 dan Tabel 5.5. Dari keseluruhan hasil uji coba skenario 1b, penggunaan RAM tertinggi pada panggilan VoIP dengan mode AES-ICM 256 pada durasi 250 detik yang mencapai 525.395Mb. Untuk nilai terendah terjadi pada mode tanpa enkripsi pada durasi 50 detik dengan 483.210Mb. Secara keseluruhan, mode tanpa enkripsi memiliki penggunaan RAM terendah, yang mencapai 12.6% pada 50 detik. Ini disebabkan

mode tanpa enkripsi tidak melakukan enkripsi pada data sehingga tidak memakai banyak RAM daripada mode enkripsi. Sedangkan penggunaan tertinggi untuk mode enkripsi adalah AES-ICM 256 pada durasi 250 detik dengan penggunaan RAM sebesar 13.7%.

Dilihat pada Tabel 5.5, banyak mode yang mencapai penggunaan RAM tertinggi terjadi pada durasi 250 detik yang menandakan panggilan VoIP pada durasi 250 banyak mengambil sumber daya RAM dan mempengaruhi performa klien. Setelah dievaluasi lebih lanjut, mode tanpa enkripsi memiliki rata-rata penggunaan RAM terendah sebesar 499.572Mb. Sedangkan untuk mode enkripsi dengan rata-rata terendah adalah AES-ICM 256 yaitu 503.919Mb.



Gambar 5.2 Grafik Penggunaan RAM pada Codec GSM

Tabel 5.5 Hasil Uji Coba Panggilan VoIP terhadap Penggunaan RAM dengan Codec GSM

Enkripsi	Durasi Panggilan				
	50s	100s	150s	200s	250s
No Encryption	**483.210**	498.550	**501.107**	**506.220**	**508.777**
AES-ICM (128 bit)	507.498	508.777	511.333	510.055	513.890
AES-ICM (256 bit)	487.045	**494.715**	502.385	510.055	**525.395**

Enkripsi	Durasi Panggilan				
	50s	100s	150s	200s	250s
AES-OFB (128 bit)	**511.333**	**511.333**	**512.612**	**512.612**	513.890
AES-OFB (256 bit)	507.498	**511.333**	511.333	511.333	513.890

5.3.1.5 Skenario Uji Coba 1c

Skenario uji coba 1c adalah pengujian panggilan VoIP dengan *codec* GSM pada klien yang mengimplementasikan mode tanpa enkripsi, enkripsi mode ICM 128, enkripsi mode ICM 256, enkripsi mode OFB 128, dan enkripsi mode OFB 256. Variasi waktu yang digunakan pada uji coba 1c adalah 50 detik, 100 detik, 150 detik, 200 detik, dan 250 detik. Dalam uji coba 1c, data yang dicatat adalah *jitter* maksimal pada saat panggilan VoIP berlangsung. Nilai tertinggi di tiap variasi waktu diberi latar belakang warna merah sedangkan nilai terendah berwarna hijau dengan tulisan berwarna putih.

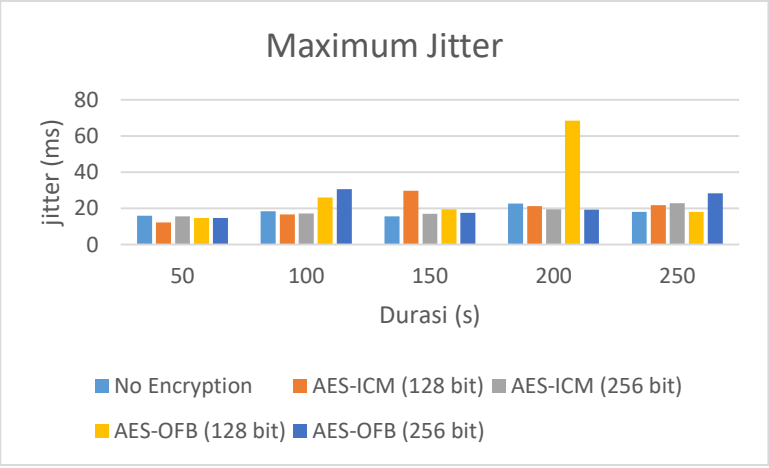
5.3.1.6 Evaluasi Uji Coba 1c

Setelah dilakukan uji coba seperti pada skenario 1c, didapatkan hasil berupa *jitter* maksimal yang ditunjukkan oleh Gambar 5.3 dan Tabel 5.6. Semakin tinggi nilai *jitter* maka kualitas dari panggilan VoIP akan semakin buruk [15]. Dapat diamati bahwa saat panggilan VoIP dengan *codec* GSM pada seluruh mode enkripsi maupun tanpa enkripsi memiliki maksimum *jitter* diatas 10 ms.

Dari keseluruhan hasil uji coba skenario 1c, nilai maksimum *jitter* pada panggilan VoIP dengan mode AES-OFB 128 pada durasi 200 detik menjadi nilai terbesar dengan 68.552 ms. Untuk nilai terendah terjadi pada mode AES-ICM 128 pada durasi 50 detik dengan 12.238 ms. Pada Tabel 5.6 dapat dilihat bahwa saat nilai maksimum *jitter* tertinggi pada AES-OFB 128 terjadi pada

saat panggilan VoIP berdurasi lama. Namun kenaikan seluruh maksimum *jitter* terjadi saat durasi 100 detik dengan mode tanpa enkripsi naik sebesar 16%, AES-ICM 128 naik sebesar 36%, AES-ICM 256 naik sebesar 9%, AES-OFB 128 naik sebesar 76% dan AES-OFB 256 naik sebesar 108%. Jadi dapat disimpulkan bahwa memasuki durasi 100 detik mengalami peningkatan nilai maksimum *jitter* sehingga yang berdampak buruk pada kualitas panggilan VoIP.

Mode AES-OFB 128 mendapat maksimum *jitter* yang rendah di awal, namun selanjutnya nilai maksimum *jitter* naik drastis sebesar 76% lalu kemudian turun 24% lalu mencapai puncak *jitter* pada durasi 200 detik dengan naik sebesar 250% yang menunjukkan mode AES-OFB 128 dengan *codec* GSM tidak stabil memasuki durasi panggilan yang meningkat. Setelah dievaluasi lebih lanjut, mode AES-OFB 256 memiliki tren maksimum *jitter* terendah dengan rata-rata kenaikan terendah yaitu 26.979 ms atau turun sebesar 5%. Untuk AES-OFB 128 memiliki tren maksimum *jitter* terburuk yaitu naik 44.196 ms atau naik 144% dibandingkan AES-ICM 128 yang naik hanya 30%.



Gambar 5.3 Grafik Maksimum Jitter pada Codec GSM

Tabel 5.6 Hasil Uji Coba Panggilan VoIP terhadap Maksimum Jitter dengan Codec GSM

Enkripsi	Durasi Panggilan				
	50s	100s	150s	200s	250s
No Encryption	**15.904**	18.432	**15.604**	22.614	18.114
AES-ICM (128 bit)	**12.238**	**16.678**	**29.796**	21.244	21.782
AES-ICM (256 bit)	15.668	17.140	17.094	19.422	22.778
AES-OFB (128 bit)	14.726	26.046	19.572	**68.552**	**18.092**
AES-OFB (256 bit)	14.702	**30.642**	17.482	**19.362**	**28.434**

5.3.1.7 Skenario Uji Coba 1d

Skenario uji coba 1d adalah pengujian panggilan VoIP dengan *codec* GSM pada klien yang mengimplementasikan mode tanpa enkripsi, enkripsi mode ICM 128, enkripsi mode ICM 256, enkripsi mode OFB 128, dan enkripsi mode OFB 256. Variasi waktu yang digunakan pada uji coba 1d adalah 50 detik, 100 detik, 150 detik, 200 detik, dan 250 detik. Dalam uji coba 1d, data yang dicatat adalah rata-rata *jitter* pada saat panggilan VoIP berlangsung. Nilai tertinggi di tiap variasi waktu diberi latar belakang warna merah sedangkan nilai terendah berwarna hijau dengan tulisan berwarna putih.

5.3.1.8 Evaluasi Uji Coba 1d

Setelah dilakukan uji coba seperti pada skenario 1d, didapatkan hasil berupa rata-rata *jitter* yang ditunjukkan oleh Gambar 5.4 dan Tabel 5.7. Semakin tinggi nilai *jitter* maka kualitas dari panggilan VoIP akan semakin buruk [15].

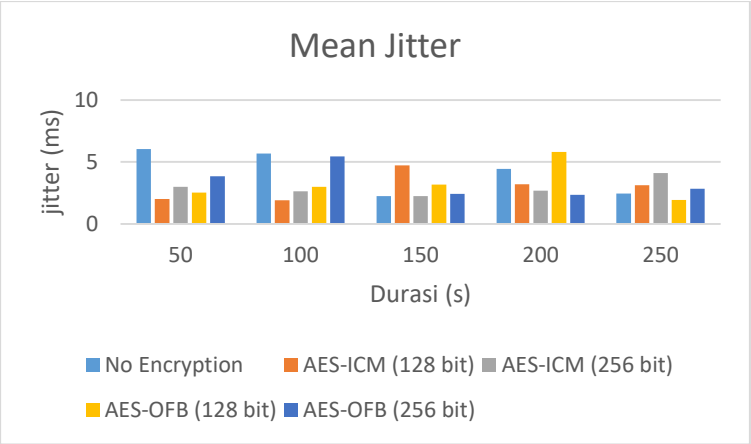
Dari keseluruhan hasil uji coba skenario 1d, nilai rata-rata *jitter* pada panggilan VoIP dengan mode tanpa enkripsi pada durasi 50 detik menjadi nilai terbesar dengan 6.046 ms. Untuk nilai terendah terjadi pada mode enkripsi AES-ICM 128 pada durasi 100

detik dengan 1.914 ms. Pada Tabel 5.7 dapat dilihat bahwa saat nilai rata-rata tertinggi pada mode tanpa enktipsi terjadi pada saat awal panggilan VoIP. Namun kenaikan rata-rata *jitter* tertinggi terjadi saat durasi 200 detik dengan AES-ICM 256 naik sebesar 19%, dan AES-OFB 128 naik sebesar 83%, dan yang terbesar mode tanpa enkripsi naik sebesar 97% sedangkan mode AES-ICM 128 mengalami penurunan sebesar 32% dan AES-OFB 256 turun sebesar 3%. Jadi dapat disimpulkan bahwa memasuki durasi 200 detik, 3 dari 5 mode pengujian mengalami peningkatan nilai rata-rata *jitter* sehingga yang berdampak buruk pada kualitas panggilan VoIP.

Mode AES-OFB 128 mendapat rata-rata *jitter* yang rendah di awal, namun selanjutnya nilai rata-rata *jitter* naik sebesar 18% pada durasi 100 detik dan lalu naik kembali sebesar 5% yang menunjukkan mode AES-OFB 128 tidak stabil memasuki durasi panggilan yang meningkat. Setelah dievaluasi lebih lanjut, AES-OFB 128 memiliki rata-rata *jitter* terbaik dengan rata-rata kenaikan yang sangat rendah yaitu turun sebesar 66%. Sedangkan AES-OFB 256 yang buruk di awal panggilan lebih baik dari AES-ICM 256 dengan kenaikan rata-rata jitter terakhir sebesar 20% berbanding 53%.

Tabel 5.7 Hasil Uji Coba Panggilan VoIP terhadap rata-rata Jitter dengan Codec GSM

Enkripsi	Durasi Panggilan				
	50s	100s	150s	200s	250s
No Encryption	**6.046**	**5.674**	2.252	4.444	2.464
AES-ICM (128 bit)	**2.010**	**1.914**	**4.722**	3.198	3.124
AES-ICM (256 bit)	3.002	2.642	**2.244**	2.686	**4.118**
AES-OFB (128 bit)	2.540	3.010	3.170	**5.820**	**1.926**
AES-OFB (256 bit)	3.860	5.442	2.418	**2.344**	2.836



Gambar 5.4 Grafik rata-rata Jitter pada Codec GSM

5.3.2 Skenario Uji Coba 2

Skenario uji coba 2 adalah pengujian panggilan VoIP dengan *codec* G722 pada klien yang mengimplementasikan mode tanpa enkripsi, enkripsi mode ICM 128, enkripsi mode ICM 256, enkripsi mode OFB 128, dan enkripsi mode OFB 256. Variasi waktu yang digunakan pada uji coba 2 adalah 50 detik, 100 detik, 150 detik, 200 detik, dan 250 detik. Data yang dicatat dan dievaluasi pada uji coba 2 adalah *traffic* data RTP yang memuat data suara dari klien, penggunaan RAM maksimal, *jitter* maksimal dan rata-rata *jitter*.

5.3.2.1 Skenario Uji Coba 2a

Skenario uji coba 2a adalah pengujian panggilan VoIP dengan *codec* G722 pada klien yang mengimplementasikan mode tanpa enkripsi, enkripsi mode ICM 128, enkripsi mode ICM 256, enkripsi mode OFB 128, dan enkripsi mode OFB 256. Variasi waktu yang digunakan pada uji coba 2a adalah 50 detik, 100 detik, 150 detik, 200 detik, dan 250 detik. Dalam uji coba 2a, data yang

dicatat adalah *traffic* dari data RTP yang memuat data suara dari klien. Nilai tertinggi di tiap variasi waktu diberi latar belakang warna merah sedangkan nilai terendah berwarna hijau dengan tulisan berwarna putih.

5.3.2.2 Evaluasi Uji Coba 2a

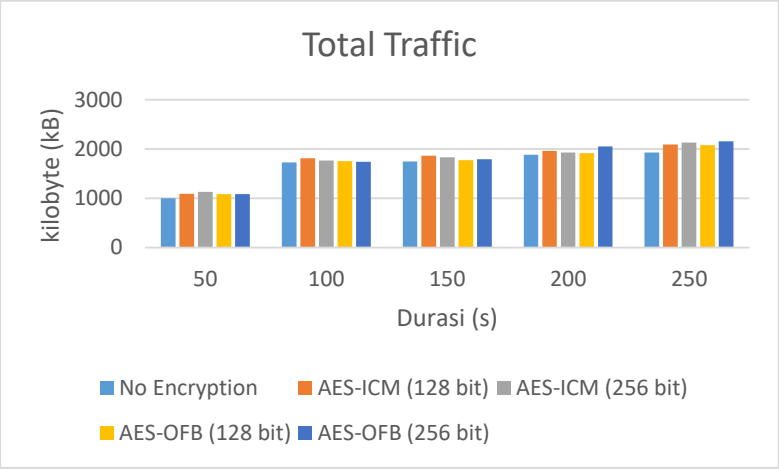
Setelah dilakukan uji coba seperti pada skenario 2a, didapatkan hasil berupa total *traffic* data RTP yang ditunjukkan oleh Gambar 5.5 dan hasil secara rinci pada Tabel 5.8. Pada Gambar 5.5 dapat diamati bahwa total *traffic* data RTP selalu meningkat atau berbanding lurus dengan penambahan durasi panggilan VoIP. Secara keseluruhan, panggilan VoIP tanpa menggunakan mode enkripsi mendapatkan total traffic RTP yang lebih kecil daripada menggunakan mode enkripsi. Ini disebabkan ukuran sebuah paket RTP dengan *codec* G722 pada mode tanpa enkripsi 214 *byte*, sedangkan dengan enkripsi 224 *byte*.

Untuk awal panggilan VoIP, terlihat AES-ICM 256 mengirim paket RTP terbesar yaitu 13% lebih banyak dari total paket RTP mode tanpa enkripsi. Pada Durasi selanjutnya yaitu 100 dan 150 detik, mode AES-ICM 128 terus mengirim paket RTP terbesar yaitu asing-masing 4% dan 6% lebih banyak dari mode tanpa enkripsi. Setelah panggilan memasuki durasi yang lama, total paket RTP didominasi oleh mode enkripsi AES-OFB 256. Pada durasi 200 detik dan 250 detik AES-OFB 256 memroses data RTP lebih banyak dengan masing-masing 9% dan 11%. Ini menunjukkan mode enkripsi pada panggilan VoIP menggunakan codec G722 memroses banyak paket RTP yang dapat memenuhi *traffic* pada jaringan.

AES-OFB 128 mengirim lebih sedikit paket SRTP dari AES-ICM 128 dari durasi awal hingga durasi terlama mengirim yang menunjukkan panggilan AES-OFB 128 tidak lebih membebani jaringan daripada AES-ICM 128. Sedangkan AES-OFB 256 mengirim lebih banyak paket SRTP dari AES-ICM 256 pada durasi lama yang menunjukkan panggilan AES-OFB 256

lebih membebani jaringan daripada AES-ICM 256 diatas 150 detik. Untuk keseluruhan mode enkripsi memiliki nilai *throughput* lebih tinggi dari *bandwidth* G722 (8 kBps) sehingga paket dapat melewati jaringan dengan baik.

Jika dilihat Tabel 5.9, tren dari *throughput* cenderung terus menurun pada seluruh mode enkripsi dari durasi 50 detik hingga durasi 250 detik. Ini disebabkan terjadinya penumpukan paket RTP pada jaringan yang membuat klien mengurangi pengiriman paket RTP sehingga dapat menyebabkan kualitas suara menurun. Walaupun terus menurun setelah 250 detik namun nilai *throughput* dari *codec* G722 lebih tinggi dari GSM sehingga dapat disimpulkan kualitas layanan *codec* G722 lebih baik dari GSM. Nilai *throughput* tertinggi adalah AES-OFB 256 sebesar 8.614 kBps dengan G722 mempengaruhi *bandwidth* sebesar 92.8% sedangkan untuk mode AES-OFB 128 lebih rendah 3% dan AES-ICM 256 lebih rendah 1.1% dengan G722 mempengaruhi *bandwidth* sebesar 96.3% dan 94%.



Gambar 5.5 Grafik Total Traffic RTP pada Codec G722

Tabel 5.8 Hasil Uji Coba Panggilan VoIP terhadap Total Traffic RTP dengan Codec G722 (kB)

Enkripsi	Durasi Panggilan				
	50s	100s	150s	200s	250s
No Encryption	**996.103**	**1726.921**	**1745.017**	**1880.556**	**1928.759**
AES-ICM (128 bit)	1087.363	**1809.281**	**1858.981**	1959.206	2091.206
AES-ICM (256 bit)	**1126.169**	1764.350	1828.050	1924.694	2129.400
AES-OFB (128 bit)	1085.306	1748.994	1770.081	1915.331	2076.703
AES-OFB (256 bit)	1080.450	1735.825	1788.500	**2050.519**	**2153.550**

Tabel 5.9 Perhitungan Throughput pada Codec G722 (kBps)

Enkripsi	Durasi Panggilan				
	50s	100s	150s	200s	250s
No Encryption	**19.922**	**17.269**	**11.633**	**9.403**	**7.715**
AES-ICM (128 bit)	21.747	**18.093**	**12.393**	9.796	8.365
AES-ICM (256 bit)	**22.523**	17.644	12.187	9.623	8.518
AES-OFB (128 bit)	21.706	17.490	11.801	9.577	8.307
AES-OFB (256 bit)	21.609	17.358	11.923	**10.253**	**8.614**

5.3.2.3 Skenario Uji Coba 2b

Skenario uji coba 2b adalah pengujian panggilan VoIP dengan *codec* G722 pada klien yang mengimplementasikan mode tanpa enkripsi, enkripsi mode ICM 128, enkripsi mode ICM 256, enkripsi mode OFB 128, dan enkripsi mode OFB 256. Variasi waktu yang digunakan pada uji coba 2b adalah 50 detik, 100 detik, 150 detik, 200 detik, dan 250 detik. Dalam uji coba 2b, data yang dicatat adalah penggunaan RAM pada saat panggilan VoIP berlangsung. Nilai tertinggi di tiap variasi waktu diberi latar belakang warna merah sedangkan nilai terendah berwarna hijau dengan tulisan berwarna putih.

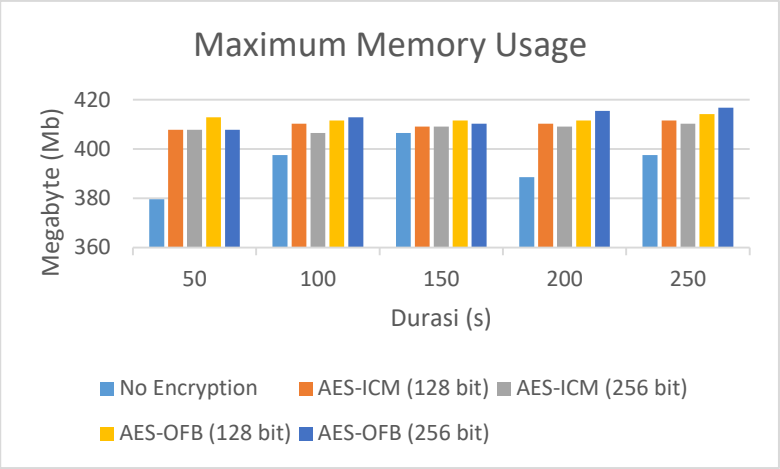
5.3.2.4 Evaluasi Uji Coba 2b

Setelah dilakukan uji coba seperti pada skenario 2b, didapatkan hasil berupa penggunaan maksimal RAM yang ditunjukkan oleh Gambar 5.6 dan Tabel 5.10. Dari keseluruhan hasil uji coba skenario 2b, penggunaan RAM tertinggi pada panggilan VoIP dengan mode AES-OFB 256 pada durasi 250 detik yang mencapai 416.737 Mb. Untuk nilai terendah terjadi pada mode tanpa enkripsi pada durasi 50 detik dengan 379.665 Mb. Secara keseluruhan, mode tanpa enkripsi memiliki penggunaan RAM terendah, yang mencapai 9.9% pada 50 detik. Ini disebabkan mode tanpa enkripsi tidak melakukan enkripsi data sehingga tidak memakai banyak RAM daripada mode enkripsi. Sedangkan penggunaan tertinggi untuk mode enkripsi adalah AES-OFB 256 pada durasi 250 detik dengan penggunaan RAM sebesar 10.9%.

Dilihat pada Tabel 5.10, banyak mode yang mencapai penggunaan RAM tertinggi terjadi pada durasi 250 detik yang menandakan panggilan VoIP durasi 250 atau lebih banyak menggunakan sumber daya RAM dan mempengaruhi performa klien. Setelah dievaluasi lebih lanjut, mode tanpa enkripsi memiliki rata-rata penggunaan RAM terendah sebesar 393.982 Mb. Sedangkan untuk mode enkripsi dengan rata-rata terendah adalah AES-ICM 256 yaitu 408.555 Mb.

Tabel 5.10 Hasil Uji Coba Panggilan VoIP terhadap Penggunaan RAM dengan Codec G722

Enkripsi	Durasi Panggilan				
	50s	100s	150s	200s	250s
No Encryption	**379.665**	**397.562**	**406.510**	**388.613**	**397.562**
AES-ICM (128 bit)	407.788	410.345	409.067	410.345	411.623
AES-ICM (256 bit)	407.788	406.510	409.067	409.067	410.345
AES-OFB (128 bit)	**412.902**	411.623	**411.623**	411.623	414.180
AES-OFB (256 bit)	407.788	**412.902**	410.345	**415.458**	**416.737**



Gambar 5.6 Grafik Penggunaan RAM pada Codec G722

5.3.2.5 Skenario Uji Coba 2c

Skenario uji coba 2c adalah pengujian panggilan VoIP dengan *codec* G722 pada klien yang mengimplementasikan mode tanpa enkripsi, enkripsi mode ICM 128, enkripsi mode ICM 256, enkripsi mode OFB 128, dan enkripsi mode OFB 256. Variasi waktu yang digunakan pada uji coba 2c adalah 50 detik, 100 detik, 150 detik, 200 detik, dan 250 detik. Dalam uji coba 2c, data yang dicatat adalah *jitter* maksimal pada saat panggilan VoIP berlangsung. Nilai tertinggi di tiap variasi waktu diberi latar belakang warna merah sedangkan nilai terendah berwarna hijau dengan tulisan berwarna putih.

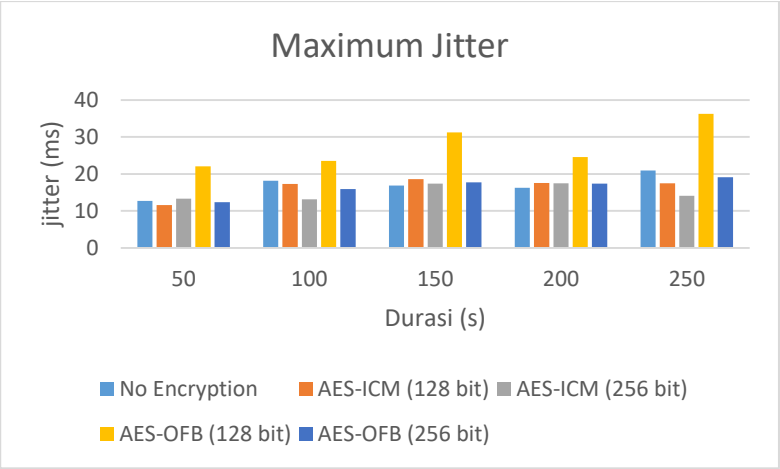
5.3.2.6 Evaluasi Uji Coba 2c

Setelah dilakukan uji coba seperti pada skenario 2c, didapatkan hasil berupa *jitter* maksimal yang ditunjukkan oleh Gambar 5.7 dan Tabel 5.11. Semakin tinggi nilai *jitter* maka

kualitas dari panggilan VoIP akan semakin buruk [15]. Dapat diamati bahwa saat panggilan VoIP dengan *codec* G722 pada seluruh mode enkripsi maupun tanpa enkripsi memiliki maksimum *jitter* diatas 10 ms.

Dari keseluruhan hasil uji coba skenario 2c, nilai maksimum *jitter* pada panggilan VoIP dengan mode AES-OFB 128 pada durasi 250 detik menjadi nilai terbesar dengan 36.256 ms. Untuk nilai terendah terjadi pada mode AES-ICM 128 pada durasi 50 detik dengan 11.594 ms. Pada Tabel 5.11 dapat dilihat bahwa saat nilai maksimum *jitter* tertinggi pada AES-OFB 128 terjadi pada saat durasi panggilan VoIP 250 detik. Namun kenaikan maksimum *jitter* terjadi saat durasi 150 detik dengan AES-ICM 128 naik sebesar 7%, AES-ICM 256 naik sebesar 32%, AES-OFB 128 naik sebesar 33% dan AES-OFB 256 naik sebesar 11% sedangkan mode tanpa enkripsi mengalami penurunan sebesar 7%. Jadi dapat disimpulkan bahwa memasuki durasi 150, 4 dari 5 mode pengujian mengalami peningkatan nilai maksimum *jitter* sehingga yang berdampak buruk pada kualitas panggilan VoIP.

Mode tanpa enkripsi mendapat maksimum *jitter* yang rendah di awal, namun selanjutnya nilai maksimum jitter naik sebesar 43% dan mengalami penurunan 7% lalu kemudian turun kembali 3% yang menunjukkan mode tanpa enkripsi dengan *codec* G722 tidak stabil memasuki durasi panggilan yang meningkat. Setelah dievaluasi lebih lanjut, AES-OFB 128 memiliki maksimum *jitter* terburuk yaitu naik 28.284 ms tapi memiliki tren maksimum *jitter* yang baik dengan turun 21% dibandingkan AES-ICM 128 yang naik 11%. Hal yang sama terjadi pada AES-OFB 256 memiliki maksimum *jitter* lebih tinggi dari AES-ICM 256 yaitu 16.767 ms tetapi memiliki tren lebih baik dengan penurunan 12% daripada AES-ICM 256 dengan memiliki tren kenaikan maksimum *jitter* 16%.



Gambar 5.7 Grafik Maksimum Jitter pada Codec G722

Tabel 5.11 Hasil Uji Coba Panggilan VoIP terhadap Maksimum Jitter dengan Codec G722

Enkripsi	Durasi Panggilan				
	50s	100s	150s	200s	250s
No Encryption	12.674	18.194	**16.872**	**16.282**	20.906
AES-ICM (128 bit)	**11.594**	17.336	18.572	17.564	17.494
AES-ICM (256 bit)	13.324	**13.108**	17.388	17.430	**14.118**
AES-OFB (128 bit)	**22.034**	**23.514**	**31.276**	**24.574**	**36.256**
AES-OFB (256 bit)	12.318	15.912	17.762	17.398	19.140

5.3.2.7 Skenario Uji Coba 2d

Skenario uji coba 2d adalah pengujian panggilan VoIP dengan *codec* G722 pada klien yang mengimplementasikan mode tanpa enkripsi, enkripsi mode ICM 128, enkripsi mode ICM 256, enkripsi mode OFB 128, dan enkripsi mode OFB 256. Variasi waktu yang digunakan pada uji coba 2d adalah 50 detik, 100 detik,

150 detik, 200 detik, dan 250 detik. Dalam uji coba 2d, data yang dicatat adalah rata-rata *jitter* pada saat panggilan VoIP berlangsung. Nilai tertinggi di tiap variasi waktu diberi latar belakang warna merah sedangkan nilai terendah berwarna hijau dengan tulisan berwarna putih.

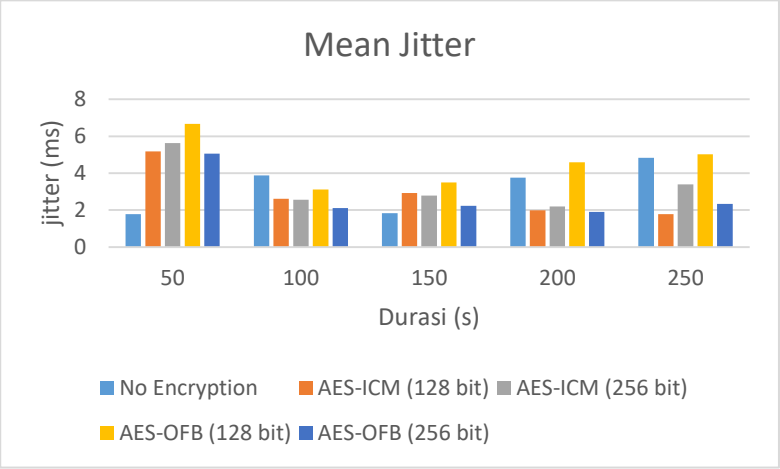
5.3.2.8 Evaluasi Uji Coba 2d

Setelah dilakukan uji coba seperti pada skenario 2d, didapatkan hasil berupa rata-rata *jitter* yang ditunjukkan oleh Gambar 5.8 dan Tabel 5.12. Semakin tinggi nilai *jitter* maka kualitas dari panggilan VoIP akan semakin buruk [15].

Dari keseluruhan hasil uji coba skenario 2d, nilai rata-rata *jitter* pada panggilan VoIP dengan mode AES-OFB 128 pada durasi 50 detik menjadi nilai terbesar dengan 6.664 ms. Untuk nilai terendah terjadi pada mode tanpa enkripsi pada durasi 50 detik dengan 1.778 ms. Pada Tabel 5.12 dapat dilihat bahwa saat nilai rata-rata tertinggi pada AES-OFB 128 terjadi pada saat awal panggilan VoIP. Namun kenaikan rata-rata *jitter* terjadi saat durasi 250 detik dengan mode tanpa enkripsi naik sebesar 28%, AES-ICM 256 naik sebesar 54%, AES-OFB 128 naik sebesar 9%, AES-OFB 256 naik sebesar 21% sedangkan AES-ICM 128 mengalami penurunan sebesar 10%. Jadi dapat disimpulkan bahwa memasuki durasi 250, 4 dari 5 mode pengujian mengalami peningkatan nilai rata-rata *jitter* sehingga yang berdampak buruk pada kualitas panggilan VoIP.

Mode AES-OFB 128 mendapat rata-rata *jitter* tertinggi di awal, namun selanjutnya nilai rata-rata *jitter* turun sebesar 53% pada durasi 100 detik dan naik kembali sebesar 12% yang menunjukkan mode AES-OFB 128 tidak stabil memasuki durasi panggilan yang meningkat. Setelah dievaluasi lebih lanjut, AES-ICM 256 memiliki rata-rata *jitter* terbaik dengan rata-rata kenaikan yang sangat rendah yaitu sebesar 2%. AES-OFB 128 memiliki rata-rata *jitter* yang lebih buruk dari AES-ICM 128 pada durasi 250 detik yaitu 5.026 ms berbanding 1.784 ms. Sedangkan AES-OFB

256 memiliki catatan lebih baik dari AES-ICM 256, AES-OFB 256 naik sebesar 21% sedangkan AES-ICM 256 turun 54%.



Gambar 5.8 Grafik rata-rata Jitter pada Codec G722

Tabel 5.12 Hasil Uji Coba Panggilan VoIP terhadap rata-rata Jitter dengan Codec G722

Enkripsi	Durasi Panggilan				
	50s	100s	150s	200s	250s
No Encryption	**1.778**	**3.882**	**1.834**	3.756	4.826
AES-ICM (128 bit)	5.186	2.616	2.934	1.992	**1.784**
AES-ICM (256 bit)	5.636	2.560	2.786	2.202	3.398
AES-OFB (128 bit)	**6.664**	3.118	**3.502**	**4.584**	**5.026**
AES-OFB (256 bit)	5.050	2.104	2.230	**1.912**	2.332

5.3.3 Skenario Uji Coba 3

Skenario uji coba 3 adalah pengujian panggilan VoIP dengan *codec* G711 pada klien yang mengimplementasikan mode tanpa enkripsi, enkripsi mode ICM 128, enkripsi mode ICM 256,

enkripsi mode OFB 128, dan enkripsi mode OFB 256. Variasi waktu yang digunakan pada uji coba 2 adalah 50 detik, 100 detik, 150 detik, 200 detik, dan 250 detik. Data yang dicatat dan dievaluasi pada uji coba 2 adalah *traffic* data RTP yang memuat data suara dari klien, penggunaan RAM maksimal, *jitter* maksimal dan rata-rata *jitter*.

5.3.3.1 Skenario Uji Coba 3a

Skenario uji coba 3a adalah pengujian panggilan VoIP dengan *codec* G711 pada klien yang mengimplementasikan mode tanpa enkripsi, enkripsi mode ICM 128, enkripsi mode ICM 256, enkripsi mode OFB 128, dan enkripsi mode OFB 256. Variasi waktu yang digunakan pada uji coba 3a adalah 50 detik, 100 detik, 150 detik, 200 detik, dan 250 detik. Dalam uji coba 3a, data yang dicatat adalah *traffic* dari data RTP yang memuat data suara dari klien. Nilai tertinggi di tiap variasi waktu diberi latar belakang warna merah sedangkan nilai terendah berwarna hijau dengan tulisan berwarna putih.

5.3.3.2 Evaluasi Uji Coba 3a

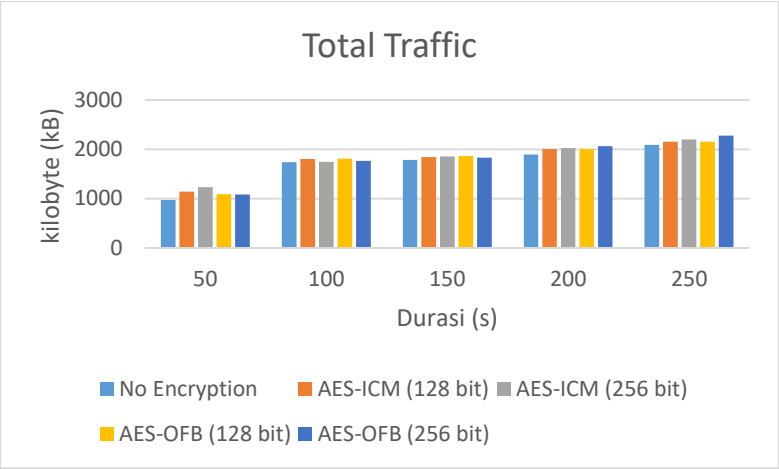
Setelah dilakukan uji coba seperti pada skenario 3a, didapatkan hasil berupa total *traffic* data RTP yang ditunjukkan oleh grafik Gambar 5.9 dan hasil secara rinci pada Tabel 5.13. Pada Gambar 5.9 dapat diamati bahwa total *traffic* data RTP selalu mengingkat atau berbanding lurus dengan penambahan durasi panggilan VoIP. Secara keseluruhan, panggilan VoIP tanpa menggunakan mode enkripsi mendapatkan total traffic RTP yang lebih kecil daripada menggunakan mode enkripsi. Ini disebabkan ukuran sebuah paket RTP dengan codec G711 pada mode tanpa enkripsi 214 *byte*, sedangkan dengan enkripsi 224 *byte*.

Untuk awal panggilan VoIP, terlihat AES-ICM 256 mengirim banyak paket RTP dengan perbedaan sangat signifikan yaitu 27% lebih banyak dari total paket RTP mode tanpa enkripsi.

Namun setelah panggilan memasuki pertengahan uji coba, total paket RTP didominasi oleh mode enkripsi AES-OFB. Pada durasi 100 dan 150 detik AES-OFB 128 memroses data RTP lebih banyak dengan masing-masing 4% dan 4.7%. Selanjutnya AES-OFB 256 pada durasi 200 dan 250 detik yaitu lebih banyak 9% dan 8%. Ini menunjukkan mode enkripsi pada panggilan VoIP menggunakan *codec* G711 memroses banyak paket RTP yang dapat memenuhi *traffic* pada jaringan.

AES-OFB 128 mengirim lebih sedikit paket SRTP dari AES-ICM 128 pada durasi awal lalu pada durasi meningkat mengirim lebih banyak paket RTP yang menunjukkan panggilan AES-OFB 128 pada durasi lama (di atas 50 detik) lebih membebani jaringan daripada AES-ICM 128. Hal yang sama terjadi pada AES-OFB 256 mengirim lebih banyak paket SRTP dari AES-ICM 256 pada durasi lama yang menunjukkan panggilan AES-OFB 256 lebih membebani jaringan daripada AES-ICM 256 diatas 150 detik. Untuk keseluruhan mode enkripsi memiliki nilai *throughput* lebih tinggi dari *bandwidth* G711 (8 kBps) sehingga paket dapat melewati jaringan dengan baik.

Jika dilihat Tabel 5.14, tren dari *throughput* cenderung terus menurun pada seluruh mode enkripsi dari durasi 50 detik hingga durasi 250 detik. Ini disebabkan terjadinya penumpukan paket RTP pada jaringan yang membuat klien mengurangi pengiriman paket RTP sehingga dapat menyebabkan kualitas suara menurun. Walaupun terus menurun setelah 250 detik namun nilai *throughput* dari *codec* G711 juga lebih tinggi dari GSM sehingga dapat disimpulkan kualitas layanan *codec* G711 lebih baik dari GSM. Sedangkan dibandingkan dengan G722, terjadi kenaikan nilai *throughput* dengan rata-rata 4.5%. Nilai *throughput* tertinggi adalah AES-OFB 256 sebesar 9.111 kBps dengan G711 mempengaruhi *bandwidth* sebesar 87.8% sedangkan untuk mode AES-OFB 128 lebih rendah 5.3% dan AES-ICM 256 lebih rendah 3.5% dengan G711 mempengaruhi *bandwidth* sebesar 92.7% dan 91%.



Gambar 5.9 Grafik Total Traffic RTP pada Codec G711

Tabel 5.13 Hasil Uji Coba Panggilan VoIP terhadap Total Traffic RTP dengan Codec G711 (kB)

Enkripsi	Durasi Panggilan				
	50s	100s	150s	200s	250s
No Encryption	**970.440**	**1740.171**	**1781.383**	**1893.102**	**2089.927**
AES-ICM (128 bit)	1141.656	1806.788	1841.700	2007.075	2155.475
AES-ICM (256 bit)	**1232.833**	1744.400	1852.813	2026.456	2197.388
AES-OFB (128 bit)	1086.356	**1810.638**	**1865.325**	2007.709	2156.088
AES-OFB (256 bit)	1082.200	1765.706	1828.313	**2065.875**	**2277.625**

Tabel 5.14 Perhitungan Throughput pada Codec G711 (kBps)

Enkripsi	Durasi Panggilan				
	50s	100s	150s	200s	250s
No Encryption	**19.409**	**17.402**	**11.876**	**9.466**	**8.360**
AES-ICM (128 bit)	22.833	18.068	12.278	10.035	8.622
AES-ICM (256 bit)	**24.657**	17.444	12.352	10.132	8.790

Enkripsi	Durasi Panggilan				
	50s	100s	150s	200s	250s
AES-OFB (128 bit)	21.727	**18.106**	**12.436**	10.039	8.624
AES-OFB (256 bit)	21.644	17.657	12.189	**10.329**	**9.111**

5.3.3.3 Skenario Uji Coba 3b

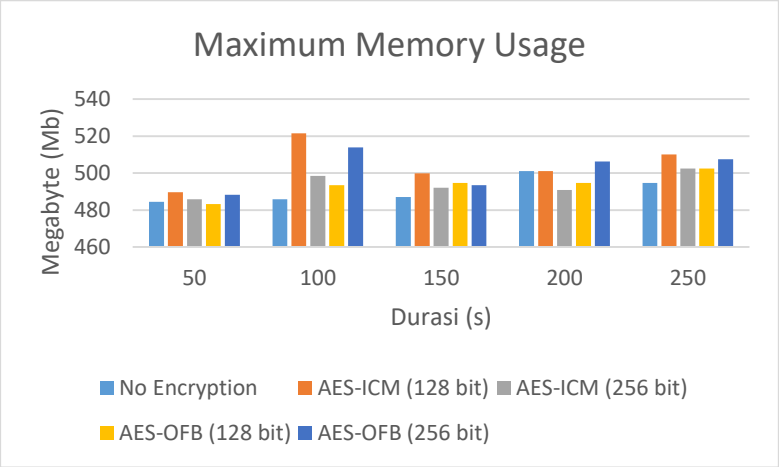
Skenario uji coba 3b adalah pengujian panggilan VoIP dengan *codec* G711 pada klien yang mengimplementasikan mode tanpa enkripsi, enkripsi mode ICM 128, enkripsi mode ICM 256, enkripsi mode OFB 128, dan enkripsi mode OFB 256. Variasi waktu yang digunakan pada uji coba 3b adalah 50 detik, 100 detik, 150 detik, 200 detik, dan 250 detik. Dalam uji coba 3b, data yang dicatat adalah penggunaan RAM pada saat panggilan VoIP berlangsung. Nilai tertinggi di tiap variasi waktu diberi latar belakang warna merah sedangkan nilai terendah berwarna hijau dengan tulisan berwarna putih.

5.3.3.4 Evaluasi Uji Coba 3b

Setelah dilakukan uji coba seperti pada skenario 3b, didapatkan hasil berupa penggunaan maksimal RAM yang ditunjukkan oleh Gambar 5.10 dan Tabel 5.15. Dari keseluruhan hasil uji coba skenario 3b, penggunaan RAM tertinggi pada panggilan VoIP dengan mode AES-ICM 128 pada durasi 100 detik yang mencapai 521.560 Mb. Untuk nilai terendah terjadi pada mode tanpa enkripsi pada durasi 50 detik dengan 483.210 Mb. Secara keseluruhan, mode tanpa enkripsi memiliki penggunaan RAM terendah, yang mencapai 12.6% pada 100 detik, 12.7% pada 150 detik dan 12.9% pada 250 detik. Ini disebabkan mode tanpa enkripsi tidak melakukan enkripsi pada data sehingga tidak memakai banyak RAM daripada mode enkripsi. Sedangkan penggunaan tertinggi untuk mode enkripsi adalah AES-ICM 128

pada durasi 100 detik dengan penggunaan 13.6% lebih banyak dari mode tanpa enkripsi.

Dilihat pada Tabel 5.15, banyak mode yang mencapai penggunaan RAM tertinggi terjadi pada awal panggilan VoIP durasi 250 detik yang menandakan panggilan VoIP pada durasi 250 banyak mengambil sumber daya RAM dan mempengaruhi performa klien. Setelah dievaluasi lebih lanjut, mode tanpa enkripsi memiliki rata-rata penggunaan RAM terendah sebesar 490.624 Mb. Sedangkan untuk mode enkripsi dengan rata-rata terendah adalah AES-OFB 128 yaitu 493.692 Mb.



Gambar 5.10 Grafik Penggunaan RAM pada Codec G711

Tabel 5.15 Hasil Uji Coba Panggilan VoIP terhadap Penggunaan RAM dengan Codec G711

Enkripsi	Durasi Panggilan				
	50s	100s	150s	200s	250s
No Encryption	484.488	**485.767**	**487.045**	501.107	**494.715**
AES-ICM (128 bit)	**489.602**	**521.560**	**499.828**	501.107	**510.055**
AES-ICM (256 bit)	485.767	498.550	492.158	**490.880**	502.385

Enkripsi	Durasi Panggilan				
	50s	100s	150s	200s	250s
AES-OFB (128 bit)	**483.210**	493.437	494.715	494.715	502.385
AES-OFB (256 bit)	488.323	513.890	493.437	**506.220**	507.498

5.3.3.5 Skenario Uji Coba 3c

Skenario uji coba 3c adalah pengujian panggilan VoIP dengan *codec* G711 pada klien yang mengimplementasikan mode tanpa enkripsi, enkripsi mode ICM 128, enkripsi mode ICM 256, enkripsi mode OFB 128, dan enkripsi mode OFB 256. Variasi waktu yang digunakan pada uji coba 3c adalah 50 detik, 100 detik, 150 detik, 200 detik, dan 250 detik. Dalam uji coba 3c, data yang dicatat adalah *jitter* maksimal pada saat panggilan VoIP berlangsung. Nilai tertinggi di tiap variasi waktu diberi latar belakang warna merah sedangkan nilai terendah berwarna hijau dengan tulisan berwarna putih.

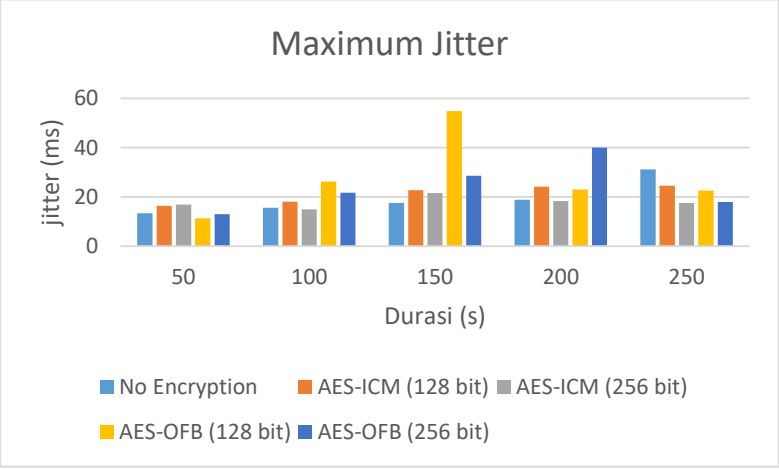
5.3.3.6 Evaluasi Uji Coba 3c

Setelah dilakukan uji coba seperti pada skenario 3c, didapatkan hasil berupa *jitter* maksimal yang ditunjukkan oleh Gambar 5.11 dan Tabel 5.16. Semakin tinggi nilai *jitter* maka kualitas dari panggilan VoIP akan semakin buruk [15]. Dapat diamati bahwa saat panggilan VoIP dengan *codec* G711 pada seluruh mode enkripsi maupun tanpa enkripsi memiliki maksimum *jitter* diatas 10 ms.

Dari keseluruhan hasil uji coba skenario 3c, nilai maksimum *jitter* pada panggilan VoIP dengan mode AES-OFB 128 pada durasi 150 detik menjadi nilai terbesar dengan 54.838 ms. Untuk nilai terendah terjadi pada mode AES-OFB 128 pada durasi 50 detik dengan 11.272 ms. Pada Tabel 5.16 dapat dilihat bahwa saat terjadi nilai maksimum tertinggi pada AES-OFB 128, terjadi

kenaikan sebesar 108% pada durasi 150 detik. Di saat yang sama, kenaikan maksimum *jitter* mode tanpa enkripsi naik sebesar 13%, AES-ICM 128 naik sebesar 26%, AES-ICM 256 naik sebesar 45%, dan AES-OFB 256 naik sebesar 31%. Jadi dapat disimpulkan bahwa memasuki durasi 150, seluruh mode pengujian mengalami peningkatan nilai maksimum *jitter* sehingga yang berdampak buruk pada panggilan VoIP.

AES-OFB 128 sempat mendapat maksimum *jitter* terendah di awal, namun selanjutnya nilai maksimum jitter naik sebesar 132% kemudian naik lagi sebesar 108% yang menunjukkan AES-OFB 128 tidak stabil memasuki durasi panggilan yang meningkat. AES-OFB 128 memiliki tren maksimum *jitter* lebih buruk dari AES-ICM 128 yaitu naik 58% dibandingkan AES-ICM 128 yang naik 4%. Hal yang sama terjadi pada AES-OFB 256 memiliki tren maksimum *jitter* lebih tinggi dari AES-ICM 256 yaitu 26.301 ms atau naik 109% daripada AES-ICM 46 dengan memiliki tren kenaikan maksimum *jitter* 11%.



Gambar 5.11 Grafik Maksimum Jitter pada Codec G711

Tabel 5.16 Hasil Uji Coba Panggilan VoIP terhadap Maksimum Jitter dengan Codec G711

Enkripsi	Durasi Panggilan				
	50s	100s	150s	200s	250s
No Encryption	13.404	15.526	**17.598**	18.830	**31.188**
AES-ICM (128 bit)	16.368	18.032	22.736	24.166	24.534
AES-ICM (256 bit)	**16.950**	**14.892**	21.590	**18.272**	**17.576**
AES-OFB (128 bit)	**11.272**	**26.246**	**54.838**	23.032	22.656
AES-OFB (256 bit)	13.040	21.672	28.578	**40.004**	17.938

5.3.3.7 Skenario Uji Coba 3d

Skenario uji coba 3d adalah pengujian panggilan VoIP dengan *codec* G711 pada klien yang mengimplementasikan mode tanpa enkripsi, enkripsi mode ICM 128, enkripsi mode ICM 256, enkripsi mode OFB 128, dan enkripsi mode OFB 256. Variasi waktu yang digunakan pada uji coba 3d adalah 50 detik, 100 detik, 150 detik, 200 detik, dan 250 detik. Dalam uji coba 3d, data yang dicatat adalah rata-rata *jitter* pada saat panggilan VoIP berlangsung. Nilai tertinggi di tiap variasi waktu diberi latar belakang warna merah sedangkan nilai terendah berwarna hijau dengan tulisan berwarna putih.

5.3.3.8 Evaluasi Uji Coba 3d

Setelah dilakukan uji coba seperti pada skenario 3d, didapatkan hasil berupa rata-rata *jitter* yang ditunjukkan oleh Gambar 5.12 dan Tabel 5.17. Semakin tinggi nilai *jitter* maka kualitas dari panggilan VoIP akan semakin buruk [15].

Dari keseluruhan hasil uji coba scenario 3d, nilai rata-rata *jitter* pada panggilan VoIP dengan mode tanpa enkripsi pada durasi 250 detik menjadi nilai terbesar dengan 6.524 ms. Untuk nilai terendah juga terjadi pada mode tanpa enkripsi pada durasi 100

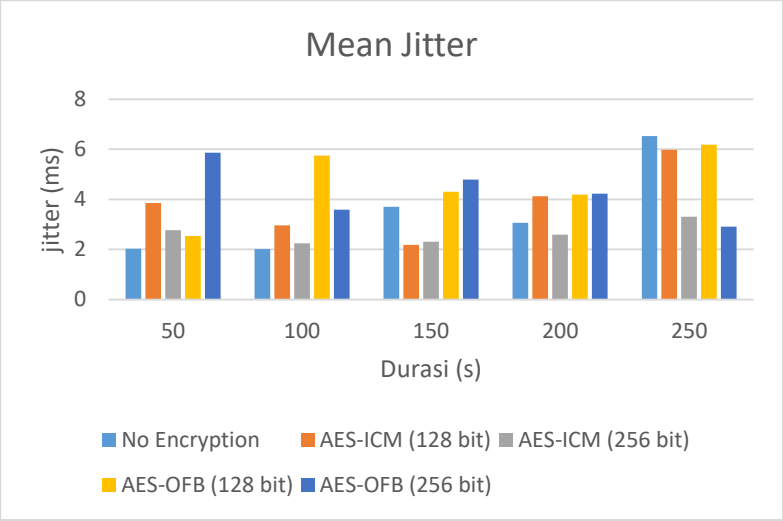
detik dengan 2.016 ms. Pada Tabel 5.17 dapat dilihat terjadi peningkatan rata-rata *jitter* pada durasi 250 detik, mode tanpa enkripsi naik sebesar 113%, AES-ICM 128 naik sebesar 44%, AES-ICM 256 naik sebesar 27%, AES-OFB 128 naik sebesar 27%, sedangkan AES-OFB 256 turun sebesar 31%. Jadi dapat disimpulkan bahwa pada durasi panggilan terlama, 4 dari 5 mode pengujian mengalami peningkatan nilai rata-rata *jitter* sehingga yang berdampak buruk pada kualitas panggilan VoIP. Beberapa faktor yang menyebabkan terjadinya naik-turun dari maksimum *jitter* dan rata-rata *jitter* adalah peningkatan lalu lintas jaringan secara tiba-tiba sehingga menimbulkan antrian paket, kecepatan klien dalam mengirim paket, dan kemampuan klien untuk menerima paket.

Mode AES-OFB 256 sempat mendapat rata-rata *jitter* tertinggi di awal, namun selanjutnya nilai rata-rata *jitter* turun sebesar 38% pada durasi 100 detik dan naik kembali sebesar 33% pada durasi 150 detik namun setelah itu terus mengalami penurunan yang menunjukkan mode AES-OFB 256 menjadi stabil memasuki durasi panggilan yang meningkat. Setelah dievaluasi lebih lanjut, AES-OFB 128 memiliki rata-rata *jitter* yang lebih buruk dari AES-ICM 128 pada durasi 250 detik yaitu 6.176 ms berbanding 5.972 ms. Berbeda dengan AES-OFB 256 memiliki catatan lebih baik dari AES-ICM 256, AES-OFB 256 dengan rata-rata 2.914 ms dengan penurunan sebesar 31% berbanding AES-ICM 256 dengan 3.310 ms yang naik sebesar 27%.

Tabel 5.17 Hasil Uji Coba Panggilan VoIP terhadap rata-rata Jitter dengan Codec G711

Enkripsi	Durasi Panggilan				
	50s	100s	150s	200s	250s
No Encryption	**2.028**	**2.016**	3.704	3.060	**6.524**
AES-ICM (128 bit)	3.858	2.956	**2.184**	4.128	5.972
AES-ICM (256 bit)	2.766	2.246	2.312	**2.586**	3.310
AES-OFB (128 bit)	2.536	**5.742**	4.308	4.182	6.176

Enkripsi	Durasi Panggilan				
	50s	100s	150s	200s	250s
AES-OFB (256 bit)	**5.864**	3.588	**4.784**	**4.228**	**2.914**



Gambar 5.12 Grafik rata-rata Jitter pada Codec G711

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan pada bab sebelumnya, yaitu Bab Uji Coba dan Evaluasi. Bab ini juga digunakan sebagai jawaban dari rumusan masalah yang dikemukakan pada Bab Pendahuluan. Selain kesimpulan, juga terdapat saran yang ditujukan untuk pengembangan penelitian lebih lanjut.

6.1 Kesimpulan

Dari hasil uji coba dan evaluasi yang dilakukan dapat diambil kesimpulan sebagai berikut:

1. Ukuran paket RTP dengan *codec* GSM pada mode enkripsi lebih besar 11% daripada mode tanpa enkripsi sedangkan paket RTP dengan *codec* G722 dan G711 pada mode enkripsi lebih besar 4.6% daripada mode tanpa enkripsi.
2. AES-OFB 128 memiliki rata-rata penggunaan RAM tertinggi pada saat panggilan VoIP yaitu 13.4% pada *codec* GSM, 10.7% pada *codec* G722, dan 12.8% pada *codec* G711.
3. AES-OFB 128 dan AES-OFB 256 memiliki nilai *throughput* yang lebih tinggi dari mode AES-ICM 128 dan AES-ICM 256 pada seluruh *codec* untuk durasi panggilan diatas 200 detik.
4. AES-OFB 128 memiliki nilai *jitter* terbaik pada durasi panggilan 50 detik menggunakan *codec* G711 dengan maksimum *jitter* 11.272 ms dan rata-rata *jitter* 2.536 ms. AES-OFB 256 memiliki nilai *jitter* terbaik pada durasi panggilan 200 detik menggunakan *codec* G722 dengan maksimum *jitter* 17.398 ms dan rata-rata *jitter* 1.912 ms.
5. AES-OFB memiliki kualitas layanan yang baik dengan *throughput* yang tinggi dari mode enkripsi lain dan nilai *jitter* di bawah 40 ms.

6.2 Saran

Saran yang diberikan untuk pengembangan aplikasi ini adalah:

1. Menggunakan media transportasi selain *User Datagram Protocol* (UDP) seperti *Transmission Control Protocol* (TCP) dan *Transport Layer Security* (TLS).
2. Pengujian dengan menggunakan tipe *codec* lainnya seperti iLBC dan G723.
3. Pengujian dengan lebih dari dua buah klien atau lebih dari sebuah sesi panggilan.
4. Membuat GUI untuk sistem klien agar lebih mudah digunakan oleh pengguna.

DAFTAR PUSTAKA

[1] M. Baugher, E. Carrara, D. A. McGrew, M. Naslund dan K. Norrman, "The Secure Real-time Transport Protocol (SRTP)," March 2004. [Online]. Available: https://tools.ietf.org/html/rfc3711. [Diakses 25 November 2016].

[2] R. Arora, "Voice over IP : Protocols and Standards," 23 November 1999. [Online]. Available: http://www.cse.wustl.edu/~jain/cis788-99/ftp/voip_protocols/. [Diakses 4 Mei 2017].

[3] M. I. Wahyuddin, "Implementasi VoIP Computer to Computer Berbasis Freeware Menggunakan Session Initiation Protocol," *ICT Research Center UNAS,* vol. 3, no. 8, pp. 50-59, 2009.

[4] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley dan E. Schooler, "SIP: Session Initiation Protocol," Academia, Juni 2002. [Online]. Available: https://www.ietf.org/rfc/rfc3261.txt. [Diakses 4 Mei 2017].

[5] CISCO, "Quality of Service (QoS)," CISCO, 2016. [Online]. Available: http://www.cisco.com/c/en/us/products/ios-nx-os-software/quality-of-service-qos/index.html. [Diakses 4 January 2017].

[6] A. A. Huurdeman, The Worldwide History of Telecommunications, New Jersey: John Wiley & Sons, Inc., 2003.

[7] ITU-T, " ITU-T Recommendation G.722," *Blue Book,* vol. III, no. 4, 1988.

[8] ITU-T, " ITU-T Recommendation G.711," *Blue Book,* vol. III, no. 4, 1988.

[9] M. J. Dworkin, E. B. Barker, J. R. Nechvatal, J. Foti, L. E. Bassham, E. Roback dan J. F. D. Jr, "ADVANCED ENCRYPTION STANDARD," *Federal Information Proceesing Standard Publication 197,* 2001.

[10] W. Stallings, Cryptography and Network Security Principles and Practice 5th Edition, New Jersey: Prentice Hall, 2011.

[11] Python Software Foundation, "About python," python.org, 2015. [Online]. Available: https://www.python.org/about/. [Diakses 4 Mei 2017].

[12] Asterisk, "Asterisk: an open source framework that lets you build communications applications for IP PBX, VoIP gateways, conference servers and custom phone apps," Asterisk, 2016. [Online]. Available: http://www.asterisk.org/. [Diakses 25 November 2016].

[13] CISCO, "Securing Internet Telephony Media with SRTP and SDP," CISCO, 2017. [Online]. Available: http://www.cisco.com/c/en/us/about/security-center/securing-voip.html#7. [Diakses 28 Mei 2017].

[14] VoIPstudio, "VoIP And How Much Jitter Is Acceptable?," VoIPstudio, 2017. [Online]. Available: https://voipstudio.com/voip-how-much-jitter-is-acceptable/. [Diakses 2 Juni 2017].

[15] L. Choo, J. Lee, H. Lee dan G. Nam, "SRMT: A Lightweight Encryption Scheme for Secure Real-time Multimedia Transmission," *International Conference on Multimedia and Ubiquitous Engineering,* pp. 60-65, 2007.

LAMPIRAN A
KODE SUMBER

Pada lampiran berikut dijelaskan mengenai kode sumber yang digunakan. Kode sumber menggunakan Bahasa Pemrograman Python, C dan Konfigurasi Asterisk. Berikut kode sumber yang dicantumkan:

1. Kode Sumber 1 Konfigurasi SIP pada Klien merupakan kode sumber dari konfigurasi Asterisk pada *file* sip.conf
2. Kode Sumber 2 Konfigurasi Extension pada Klien merupakan kode sumber dari konfigurasi Asterisk pada *file* extensions.conf
3. Kode Sumber 3 Definisi Class SIP Klien merupakan definisi dari class Callback dan MyCallCallback yang mengatur akun klien dan panggilan VoIP.
4. Kode Sumber 4 Inisialisasi PJSIP merupakan kode sumber inisialisasi dari *library* PJSIP, pengaturan akun SIP klien, dan pengaturan SRTP.
5. Kode Sumber 5 Perulangan Aktivitas SIP Klien merupakan kode sumber perulangan aktivitas klien yaitu *answer*, *hang up*, *make call*, dan *quit*.
6. Kode Sumber 6 Implementasi AES-OFB pada AES-ICM merupakan implementaasi bagian perulangan blok *cipher* AES-OFB pada AES-ICM.

Kode Sumber 1 Konfigurasi SIP pada Klien

```
1.  [7001]
2.  type=friend
3.  callerid="Asterisk1" <7001>
```

```
host=dynamic
canreinvite=no
secret=123456
dtmfmode=rfc2833
context=default
qualify=yes
disallow=all
allow=ulaw
allow=alaw
allow=gsm
allow=g722
transport=udp
encryption=yes

[7002]
type=friend
callerid="Asterisk1" <7002>
host=dynamic
canreinvite=no
secret=123456
dtmfmode=rfc2833
context=default
qualify=yes
disallow=all
```

```
28. allow=ulaw
29. allow=alaw
30. allow=gsm
31. allow=g722
32. transport=udp
33. encryption=yes
```

Kode Sumber 2 Konfigurasi Extension pada Klien

```
1.  exten => 7001,1,Answer()
2.  exten => 7001,2,Dial(SIP/7001,60)
3.  exten => 7001,3,Playback(vm-nobodyavail)
4.  exten => 7001,4,VoiceMail(7001@main)
5.  exten => 7001,5,Hangup()
6.  exten => 7001,n,Set(_SIP_SRTP_SDES=1)
7.  exten => 7001,n,Set(_SIPSRTP=optional)
8.  exten => 7001,n,Set(_SIPSRTP_CRYPTO=enable)
9.
10. exten => 7002,1,Answer()
11. exten => 7002,2,Dial(SIP/7002,60)
12. exten => 7002,3,Playback(vm-nobodyavail)
13. exten => 7002,4,VoiceMail(7002@main)
14. exten => 7002,5,Hangup()
```

```
exten => 7002,n,Set(_SIP_SRTP_SDES=1)
exten => 7002,n,Set(_SIPSRTP=optional)
exten => 7002,n,Set(_SIPSRTP_CRYPTO=enable)
```

Kode Sumber 3 Definisi Class SIP Klien

```
import sys
import pjsua as pj
import wave
from time import sleep

LOG_LEVEL=3
current_call = None
# Logging callback
def log_cb(level, str, len):
    print str,
# Callback to receive events from account
class MyAccountCallback(pj.AccountCallback):
    def __init__(self, account=None):
        pj.AccountCallback.__init__(self, account)
    # Notification on incoming call
    def on_incoming_call(self, call):
        global current_call
```

```
18.         if current_call:
19.             call.answer(486, "Busy")
20.             return
21.
22.         print "Incoming call from ", call.info().remote_uri
23.         print "Press 'a' to answer"
24.         current_call = call
25.         call_cb = MyCallCallback(current_call)
26.         current_call.set_callback(call_cb)
27.         current_call.answer(180)
28.
29. # Callback to receive events from Call
30. class MyCallCallback(pj.CallCallback):
31.     def __init__(self, call=None):
32.         pj.CallCallback.__init__(self, call)
33.     # Notification when call state has changed
34.     def on_state(self):
35.         global current_call
36.         print "Call with", self.call.info().remote_uri,
37.         print "is", self.call.info().state_text,
38.         print "last code =", self.call.info().last_code,
39.         print "(" + self.call.info().last_reason + ")"
40.
41.         if self.call.info().state == pj.CallState.DISCONNECTED:
```

```
42.             current_call = None
43.             print 'Current call is', current_call
44.
45.     # Notification when call's media state has changed.
46.     def on_media_state(self):
47.         if self.call.info().media_state == pj.MediaState.ACTIVE:
48.             # Connect the call to sound device
49.             call_slot = self.call.info().conf_slot
50.             #
51. self.wav_player_id=pj.Lib.instance().create_player('virgoun.wav',loop=False)
52.             #
53. self.wav_slot=pj.Lib.instance().player_get_slot(self.wav_player_id)
54.             # print "id wav " + str(self.wav_player_id)
55.             # print "id wav slot " + str(self.wav_slot)
56.             pj.Lib.instance().conf_connect(call_slot, 0)
57.             pj.Lib.instance().conf_connect(0, call_slot)
58.             #sleep(time)
59.             #pj.Lib.instance().player_destroy(self.wav_player_id)
60.             #self.call.hangup()
61.             #in_call = False
62.             # pj.Lib.instance().conf_connect(self.wav_slot, call_slot)
63.             # pj.Lib.instance().conf_connect(call_slot, self.wav_slot)
64.             print "Media is now active"
65.         else:
```

```
66.             print "Media is inactive"
67. # Function to make call
68. def make_call(uri):
69.     try:
70.         print "Making call to", uri
71.         return acc.make_call(uri, cb=MyCallCallback())
72.     except pj.Error, e:
73.         print "Exception: " + str(e)
74.         return Nonetransport=udp
```

Kode Sumber 4 Inisialisasi PJSIP

```
1.  # Create library instance
2.  lib = pj.Lib()
3.  try:
4.      # Init library with default config and some customized
5.      # logging config.
6.      lib.init(log_cfg = pj.LogConfig(level=LOG_LEVEL, callback=log_cb))
7.      # Create UDP transport
8.      transport = lib.create_transport(pj.TransportType.UDP,
9.                                       pj.TransportConfig(0))
10.     print "\nListening on", transport.info().host,
11.     print "port", transport.info().port, "\n"
12.
```

```
13.     # Start the library
14.     lib.start()
15.
16.     # Create local account
17.     # input server
18.     server = "10.151.34.221"
19.     user=raw_input("Enter Username: ")
20.     password = "123456"
21.
22.     acc_conf = pj.AccountConfig(domain = server, username = user, password =
23. password, display = user)
24.
25.     acc_conf.id ="sip:"+user
26.     acc_conf.reg_uri ='sip:'+server+':'+str(transport.info().port)
27.     acc_conf.use_srtp = 2
28.     acc_conf.srtp_secure_signaling = 0
29.     acc_callback = MyAccountCallback(acc_conf)
30.     acc = lib.create_account(acc_conf,cb=acc_callback)
31.
32.     # creating instance AccountCallback class
33.     acc.set_callback(acc_callback)
34.
35.     print('\n')
36.     print "Registration Complete-----------"
```

```
37.     print('Status= ',acc.info().reg_status, \
38.          '(' + acc.info().reg_reason + ')')
39.
40.     # If argument is specified then make call to the URI
41.     if len(sys.argv) > 1:
42.         lck = lib.auto_lock()
43.         current_call = make_call(sys.argv[1])
44.         print 'Current call is', current_call
45.         del lck
46.     my_sip_uri = acc_conf.id + "@" + transport.info().host + \
47.                  ":" + str(transport.info().port)
```

Kode Sumber 5 Perulangan Aktivitas SIP Klien

```
1.  # Menu loop
2.      while True:
3.          print "My SIP URI is", my_sip_uri
4.          print "Menu:  m=make call, h=hangup call, a=answer call, q=quit"
5.          input = sys.stdin.readline().rstrip("\r\n")
6.          if input == "m":
7.              if current_call:
8.                  print "Already have another call"
9.                  continue
10.             print "Enter destination URI to call: ",
```

```
11.             input = sys.stdin.readline().rstrip("\r\n")
12.             if input == "":
13.                 continue
14.             lck = lib.auto_lock()
15.             current_call = make_call(input)
16.             del lck
17.         elif input == "h":
18.             if not current_call:
19.                 print "There is no call"
20.                 continue
21.             current_call.hangup()
22.         elif input == "a":
23.             if not current_call:
24.                 print "There is no call"
25.                 continue
26.             current_call.answer(200)
27.         elif input == "q":
28.             break
29.     # Shutdown the library
30.     transport = None
31.     acc.delete()
32.     acc = None
33.     lib.destroy()
34.     lib = None
```

```
except pj.Error, e:
    print "Exception: " + str(e)
    lib.destroy()
    lib = None
```

Kode Sumber 6 Implementasi AES-OFB pada AES-ICM

```
static inline void
aes_icm_advance_ismacryp(aes_icm_ctx_t *c, uint8_t forIsmacryp) {

  int count = 0;
  //Temp of the key stream
  v128_t iv_aes = c->keystream_buffer;
  v128_t *pointer_iv_aes = &iv_aes;
  /* fill buffer with new keystream */
  v128_copy(&c->keystream_buffer, &c->counter);

  //OFB Implementation 32-bits
  while(count < 4){
    aes_encrypt(pointer_iv_aes, &c->expanded_key);
    c->keystream_buffer.v32[count] = iv_aes.v32[count];
    count++;
  }

```

```
18.    //aes_encrypt(&c->keystream_buffer, &c->expanded_key);
19.    c->bytes_in_buffer = sizeof(v128_t);
20.
21.    debug_print(mod_aes_icm, "counter:    %s",
22.                v128_hex_string(&c->counter));
23.    debug_print(mod_aes_icm, "ciphertext: %s",
24.                v128_hex_string(&c->keystream_buffer));
25.
26.    /* clock counter forward */
27.
28.    if (forIsmacryp) {
29.      uint32_t temp;
30.      //alex's clock counter forward
31.      temp = ntohl(c->counter.v32[3]);
32.      ++temp;
33.      c->counter.v32[3] = htonl(temp);
34.    } else {
35.      if (!++(c->counter.v8[15]))
36.        ++(c->counter.v8[14]);
37.    }
38.  }
```

BIODATA PENULIS

I Putu Dwi Pratama Arijaya, lahir pada tanggal 4 Maret 1995 di Denpasar. Penulis merupakan seorang mahasiswa yang sedang menempuh studi di Jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember. Memiliki beberapa hobi antara lain membaca novel, futsal, dan desain. Selama menempuh pendidikan di kampus, penulis juga aktif dalam organisasi kemahasiswaan, antara lain Staff Departemen Media Informasi Himpunan Mahasiswa Teknik Computer-Informatika dan Staff Departemen Komunikasi dan Informasi Tim Pembina Kerohanian Hindu ITS pada tahun ke-2, serta Staff Ahli Departemen Media Informasi Himpunan Mahasiswa Teknik Computer-Informatika dan Wakil Kepala Departemen Komunikasi dan Informasi Tim Pembina Kerohanian Hindu ITS pada tahun ke-3.

Kritik dan saran sangat diharapkan guna peningkatan kualitas dan penulisan selanjutnya. Untuk itu, silahkan kirim kritik dan saran ke : **arijayadwi@gmail.com**